

Network UPS Tools User Manual

Russell Kroll, Arnaud Quette, Arjen de Korte and Jim Klimov

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
2.8.3	2025-04-07	Some changes to docs and recipes, libusbclient API and functionality. Updated NUT SEMVER definition and added scripting around it. Groundwork for vendor-defined status and INSTCMD buzzwords like "ECO". Fixed some regressions and added improvements for certain new device series.	JK
2.8.2	2024-04-01	Some changes to docs and recipes, libnutscan API and functionality. Added nutconf (library and tool). Fixed some regressions and added improvements for certain new device series.	JK
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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
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
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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
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	Introduction	1
2	Network UPS Tools Overview	1
2.1	Description	1
2.2	NUT and the ecosystem	1
2.2.1	NUT Support Policy	2
2.3	Installing	2
2.4	Upgrading	2
2.5	Configuring and using	2
2.6	Documentation	2
2.7	Network Information	3
2.8	Manifest	3
2.9	Drivers	3
2.9.1	Extra Settings	4
2.9.2	Hardware Compatibility List	4
2.9.3	Generic Device Drivers	4
2.9.4	UPS Shutdowns	5
2.9.5	Power distribution unit management	5
2.10	Network Server	5
2.11	Monitoring client	5
2.11.1	Primary	6
2.11.2	Secondary	6
2.11.3	Additional Information	6
2.12	Clients	6
2.12.1	upsc	6
2.12.2	upslog	7
2.12.3	upsrw	7
2.12.4	upscmd	7
2.13	CGI Programs	8
2.13.1	Access Restrictions	8
2.13.2	upsstats	8
2.13.3	upsimage	8
2.13.4	upsset	8
2.14	Version Numbering	9
2.15	Backwards and Forwards Compatibility	9
2.16	Support / Help / etc.	9
2.17	Hacking / Development Info	9
2.18	Acknowledgements / Contributions	9

3	Features	11
3.1	Multiple manufacturer and device support	11
3.2	Multiple architecture support	11
3.3	Layered and modular design with multiple processes	11
3.4	Redundancy support — Hot swap/high availability power supplies	12
3.5	Security and access control	12
3.6	Web-based monitoring	12
3.7	Free software	12
3.8	UPS management and control	12
3.9	Monitoring diagrams	13
3.9.1	"Simple" configuration	13
3.9.2	"Advanced" configuration	13
3.9.3	"Big Box" configuration	14
3.9.4	"Bizarre" configuration	14
3.10	Image credits	15
3.11	Compatibility information	15
3.11.1	Hardware	15
3.11.2	Operating systems	15
3.11.3	NUT Support Policy	16
4	Download information	16
4.1	Source code	16
4.1.1	Stable tree: 2.8	17
4.1.2	Development tree:	17
	Code repository	17
	Browse code	17
	Snapshots	17
4.1.3	Older versions	18
4.2	Binary packages	18
4.3	Java packages	19
4.4	Virtualization packages	19
4.4.1	VMware	19
5	Installation instructions	19
5.1	Installing from source	19
5.1.1	Prepare your system	20
	System User creation	20
5.1.2	Build and install	20
	Configuration	20

Build the programs	21
Installation	21
State path creation	22
Ownership and permissions	22
5.2 Building NUT for in-place upgrades or non-disruptive tests	22
5.2.1 Overview	22
Pre-requisites	23
Getting the right sources	23
5.2.2 Testing with CI helper	23
5.2.3 Replacing a NUT deployment	24
Replacing any NUT deployment	24
Replacing a systemd-enabled NUT deployment	25
Iterating with a systemd deployment	26
5.2.4 Next steps after an in-place upgrade	26
5.3 Installing from packages	26
5.3.1 Debian, Ubuntu and other derivatives	26
5.3.2 Mandriva	27
5.3.3 SUSE / openSUSE	27
5.3.4 Red Hat, Fedora and CentOS	28
5.3.5 FreeBSD	28
Binary package	28
Port	28
USB UPS on FreeBSD	28
5.3.6 Windows	29
Windows binary package	29
Building for Windows	29
5.3.7 Runtime configuration	30
6 Configuration notes	30
6.1 Details about the configuration files	30
6.1.1 Generalities	30
6.1.2 Line spanning	31
6.2 Basic configuration	31
6.2.1 Driver configuration	32
6.2.2 Starting the driver(s) in legacy operating systems	32
6.2.3 Driver(s) as a service	33
6.2.4 Data server configuration (upsd)	34
6.2.5 Starting the data server	34
6.2.6 Check the UPS data	35

Status data	35
All data	35
6.2.7 Startup scripts	36
6.3 Configuring automatic shutdowns for low battery events	37
6.3.1 Shutdown design	37
6.3.2 How you set it up	38
NUT user creation	38
Reloading the data server	38
Power Off flag file	39
Securing upsmon.conf	39
Create a MONITOR directive for upsmon	39
Define a SHUTDOWNCMD for upsmon	40
Start upsmon	40
Checking upsmon	40
Startup scripts	40
Shutdown scripts	41
Testing shutdowns	41
6.3.3 Using suspend to disk	42
6.3.4 RAID warning	42
6.4 Typical setups for enterprise networks and data rooms	42
6.5 Typical setups for big servers with UPS redundancy	44
6.5.1 Example configuration	44
6.5.2 Multiple UPS shutdowns ordering	44
6.5.3 Other redundancy configurations	45
7 Advanced usage and scheduling notes	45
7.1 The simple approach, using your own script	45
7.1.1 How it works relative to upsmon	45
7.1.2 Setting up everything	46
7.1.3 Using more advanced features	46
7.1.4 Suppressing notify storms	46
7.2 The advanced approach, using upssched	46
7.2.1 How upssched works relative to upsmon	47
7.2.2 Setting up your upssched.conf	47
The big picture	47
Establishing timers	47
Executing commands immediately	48
7.2.3 Writing the command script handler	48
7.2.4 Early Shutdowns	48
7.2.5 Background	49

8	NUT outlets management and PDU notes	49
8.1	Introduction	49
8.2	NUT outlet data collection	50
8.3	Outlets on PDU	50
8.4	Outlets on UPS	50
8.5	Other type of devices	51
9	NUT daisychain support notes	51
9.1	Introduction	51
9.2	Implementation notes	51
9.2.1	General specification	51
	Devices status handling	51
	Devices alarms handling	52
	Example	52
9.2.2	Information for developers	52
	Base support	53
	Templates with multiple definitions	53
	Devices alarms handling	53
10	Notes on securing NUT	54
10.1	How to verify the NUT source code signature	54
10.2	How to verify the NUT source code checksum	55
10.3	System level privileges and ownership	56
10.4	NUT level user privileges	56
10.5	Network access control	56
10.5.1	NUT LISTEN directive	56
10.5.2	Firewall	57
	Uncomplicated Firewall (UFW) support	57
10.5.3	TCP Wrappers	58
10.6	Configuring SSL	58
10.6.1	OpenSSL backend usage	58
	Install OpenSSL	58
	Recompile and install NUT	58
	Create a certificate and key for upsd	58
	Figure out the hash for the key	59
	Install the client-side certificate	59
	Create the combined file for upsd	59
	Note on certification authorities (CAs) and signed keys	59
	Install the server-side certificate	60

Clean up the temporary files	60
Restart upsd	60
Point upsmon at the certificates	60
Recommended: make upsmon verify all connections with certificates	60
Recommended: force upsmon to use SSL	60
10.6.2 NSS backend usage	61
Install NSS	61
Recompile and install NUT	61
Create certificate and key for the host	61
Create a self-signed CA certificate	61
Install the server-side certificate	62
upsd (required): certificate database and self certificate	62
upsd (optional): client authentication	62
upsmon (required): upsd authentication	62
upsmon (optional): certificate database and self certificate	63
10.6.3 Restart upsd	63
10.6.4 Restart upsmon	63
10.6.5 Recommended: sniff the connection to see it for yourself	63
10.6.6 Potential problems	64
10.6.7 Conclusion	64
10.7 chrooting and other forms of paranoia	64
10.7.1 Generalities	64
10.7.2 symlinks	65
10.7.3 upsmon	65
10.7.4 Config files	66
A Glossary	66
B Acknowledgements / Contributions	66
B.1 The NUT Team	66
B.1.1 Active members	66
B.1.2 Retired members	67
B.2 Supporting manufacturers	68
B.2.1 UPS manufacturers	68
B.2.2 Appliances manufacturers	68
B.3 Other contributors	69
B.4 Older entries (before 2005)	69

C	NUT command and variable naming scheme	69
C.1	Structured naming	70
C.2	Numeric format	70
C.3	Time and Date format	71
C.4	Variables	71
C.4.1	device: General unit information	71
C.4.2	ups: General unit information	72
C.4.3	input: Incoming line/power information	74
C.4.4	output: Outgoing power/inverter information	76
C.4.5	Three-phase additions	77
	Phase Count Determination	77
	DOMAINs	77
	Specification (SPEC)	77
	CONTEXT	77
	Valid CONTEXTs	78
	Valid SPECs	78
C.4.6	EXAMPLES	79
C.4.7	battery: Any battery details	79
C.4.8	ambient: Conditions from external probe equipment	81
C.4.9	outlet: Smart outlet management	82
	outlet.group: groups of smart outlets	83
C.4.10	driver: Internal driver information	84
C.4.11	server: Internal server information	84
C.5	Instant commands	84
C.5.1	Experimental instant commands	85
D	Hardware Compatibility List	86
E	Documentation	86
E.1	User Documentation	86
E.2	Developer Documentation	86
E.3	Data dumps for the DDL	87
E.4	Offsite Links	87
E.5	News articles and Press releases	88
F	Support instructions	88
F.1	Documentation	88
F.2	Mailing lists	88
F.2.1	Request help	88
F.2.2	Post a patch, ask a development question, ...	89
F.2.3	Discuss packaging and related topics	89
F.3	IRC (Internet Relay Chat)	89
F.4	GitHub Issues	89

G	Cables information	89
G.1	APC	89
G.1.1	940-0024C clone	89
G.1.2	940-0024E clone	90
G.1.3	940-0024C clone for Macs	90
G.2	Belkin	90
G.2.1	OmniGuard F6C***-RKM	90
G.3	Eaton	91
G.3.1	MGE Office Protection Systems	91
	DB9-DB9 cable (ref 66049)	91
	DB9-RJ45 cable	92
	NMC DB9-RJ45 cable	93
	USB-RJ45 cable	93
	DB9-RJ12 cable	94
G.3.2	Powerware LanSafe	96
G.3.3	SOLA-330	96
G.4	HP - Compaq	97
G.4.1	Older Compaq UPS Family	97
G.5	Phoenixtec (Best Power)	97
G.6	Tripp-Lite	98
H	Configure options	98
H.1	Keeping a report of NUT configuration	99
H.2	In-place replacement defaults	99
H.3	Driver selection	100
H.3.1	Serial drivers	100
H.3.2	USB drivers	100
H.3.3	SNMP drivers	100
H.3.4	XML drivers and features	101
H.3.5	LLNC CHAOS Powerman driver	101
H.3.6	IPMI drivers	101
H.3.7	I2C bus drivers	101
H.3.8	GPIO bus drivers	101
H.3.9	Modbus drivers	101
H.3.10	Manual selection of drivers	102
H.4	Optional features	102
H.4.1	CGI client interface	102
H.4.2	NUT Scanner tool	102
H.4.3	NUT Configuration tool	102

H.4.4	Pretty documentation and man pages	103
H.4.5	Python support	104
H.4.6	Development files	105
H.4.7	Options for developers	106
H.4.8	I want it all!	107
H.4.9	Networking transport security	107
H.4.10	Networking access security	107
H.4.11	Networking IPv6	107
H.4.12	AVAHI/mDNS	107
H.4.13	LibLTDL	108
H.5	Other configuration options	108
H.5.1	NUT data server port	108
H.5.2	Daemon user accounts	108
H.5.3	Syslog facility	108
H.5.4	External API integration scripts	109
H.6	Installation directories	109
H.7	Directories used by NUT at run-time	112
H.8	Things the compiler might need to find	112
H.8.1	pkg-config tooling	112
H.8.2	LibGD	113
H.8.3	LibUSB	113
H.8.4	Various	113
I	Upgrading notes	113
I.1	Changes from 2.8.2 to 2.8.3	114
I.2	Changes from 2.8.1 to 2.8.2	117
I.3	Changes from 2.8.0 to 2.8.1	117
I.4	Changes from 2.7.4 to 2.8.0	119
I.5	Changes from 2.7.3 to 2.7.4	121
I.6	Changes from 2.7.2 to 2.7.3	121
I.7	Changes from 2.7.1 to 2.7.2	121
I.8	Changes from 2.6.5 to 2.7.1	121
I.9	Changes from 2.6.4 to 2.6.5	122
I.10	Changes from 2.6.3 to 2.6.4	122
I.11	Changes from 2.6.2 to 2.6.3	122
I.12	Changes from 2.6.1 to 2.6.2	122
I.13	Changes from 2.6.0 to 2.6.1	122
I.14	Changes from 2.4.3 to 2.6.0	122
I.15	Changes from 2.4.2 to 2.4.3	122

I.16	Changes from 2.4.1 to 2.4.2	122
I.17	Changes from 2.4.0 to 2.4.1	122
I.18	Changes from 2.2.2 to 2.4.0	123
I.19	Changes from 2.2.1 to 2.2.2	123
I.20	Changes from 2.2.0 to 2.2.1	123
I.21	Changes from 2.0.5 to 2.2.0	123
I.22	Changes from 2.0.4 to 2.0.5	124
I.23	Changes from 2.0.3 to 2.0.4	124
I.24	Changes from 2.0.2 to 2.0.3	124
I.25	Changes from 2.0.1 to 2.0.2	124
I.26	Changes from 2.0.0 to 2.0.1	124
I.27	Changes from 1.4.0 to 2.0.0	124
I.28	File trimmed here on changes from 1.2.2 to 1.4.0	125
J	Project history	125
J.1	Prototypes and experiments	125
J.1.1	May 1996: early status hacks	125
J.1.2	January 1997: initial protocol tests	125
J.1.3	September 1997: first client/server code	126
J.2	Smart UPS Tools	126
J.2.1	March 1998: first public release	126
J.2.2	June 1999: Redesigned, rewritten	126
J.3	Network UPS Tools	127
J.3.1	September 1999: new name, new URL	127
J.3.2	June 2001: common driver core	127
J.3.3	May 2002: casting off old drivers, IANA port, towards 1.0	127
J.4	Leaving 0.x territory	128
J.4.1	August 2002: first stable tree: NUT 1.0.0	128
J.4.2	November 2002: second stable tree: NUT 1.2.0	128
J.4.3	April 2003: new naming scheme, better driver glue, and an overhauled protocol	128
J.4.4	July 2003: third stable tree: NUT 1.4.0	129
J.4.5	July 2003: pushing towards 2.0	129
J.5	Backwards and Forwards Compatibility (NUT v1.x vs. v2.x)	129
J.6	networkupstools.org	130
J.6.1	November 2003: a new URL	130
J.7	Second major version	130
J.7.1	March 2004: NUT 2.0.0	130
J.8	The change of leadership	130
J.8.1	February 2005: NUT 2.0.1	130

1 Introduction

The primary goal of the Network UPS Tools (NUT) project is to provide support for Power Devices, such as Uninterruptible Power Supplies, Power Distribution Units and Solar Controllers.

NUT provides many control and monitoring [features](#), with a uniform control and management interface.

More than 170 different manufacturers, and several thousands of models are [compatible](#).

This software is the combined effort of many [individuals and companies](#).

This document intend to describe how to install software support for your [Power Devices](#) (UPS, PDU, ...), and how to use the NUT project. It is not intended to explain what are, nor distinguish the different technologies that exist. For such information, have a look at the [General Power Devices Information](#).

If you wish to discover how everything came together, have a look at the [Project History](#).

2 Network UPS Tools Overview

2.1 Description

Network UPS Tools is a collection of programs which provide a common interface for monitoring and administering UPS, PDU and SCD hardware. It uses a layered approach to connect all of the parts.

Drivers are provided for a wide assortment of equipment. They understand the specific language of each device and map it back to a compatibility layer. This means both an expensive high end UPS, a simple "power strip" PDU, or any other power device can be handled transparently with a uniform management interface.

This information is cached by the network server `upsd`, which then answers queries from the clients. `upsd` contains a number of access control features to limit the abilities of the clients. Only authorized hosts may monitor or control your hardware if you wish. Since the notion of monitoring over the network is built into the software, you can hang many systems off one large UPS, and they will all shut down together. You can also use NUT to power on, off or cycle your data center nodes, individually or globally through PDU outlets.

Clients such as `upsmmon` check on the status of the hardware and do things when necessary. The most important task is shutting down the operating system cleanly before the UPS runs out of power. Other programs are also provided to log information regularly, monitor status through your web browser, and more.

2.2 NUT and the ecosystem

NUT comes pre-packaged for many operating systems and embedded in storage, automation or virtualization appliances, and is also often shipped as the software companion by several UPS vendors. Of course, it is quite normal and supported to build your own — whether for an operating system which lacks it yet, or for an older distribution which lacks the current NUT version; whether to take advantage of new features or to troubleshoot a new UPS deployment with a debugger in hand.

Given its core position at the heart of your systems' lifecycle, we make it a point to have current NUT building and running anywhere, especially where older releases did work before (including "abandonware" like the servers and OSes from the turn of millennium): if those boxes are still alive and in need of power protection, they should be able to get it.

Tip

If you like how the NUT project helps protect your systems from power outages, please consider sponsoring or at least "starring" it on GitHub at <https://github.com/networkupstools/nut/> - these stars are among metrics which the larger potential sponsors consider when choosing how to help FOSS projects. Keeping the lights shining in such a large non-regression build matrix is a big undertaking!

See [acknowledgements of organizations which help with NUT CI and other daily operations](#) for an overview of the shared effort.

As a FOSS project, for over a quarter of a century we welcome contributions of both core code (drivers and other features), build recipes and other integration elements to make it work on your favourite system, documentation revisions to make it more accessible to newcomers, as well as hardware vendor cooperation with first-hand driver and protocol submissions, and just about anything else you can think of.

2.2.1 NUT Support Policy

The Network UPS Tools project is a community-made open-source effort, primarily made and maintained by people donating their spare time.

The support channels are likewise open, with preferred ones being [the NUT project issue tracker on GitHub](#) and the NUT Users mailing list, as detailed at <https://networkupstools.org/support.html> page.

Please keep in mind that any help is provided by community members just like yourself, as a best effort, and subject to their availability and experience. It is expected that you have read the Frequently Asked Questions, looked at the [NUT wiki](#), and have a good grasp about the three-layer design and programs involved in a running deployment of NUT, for a discussion to be constructive and efficient.

Be patient, polite, and prepare to learn and provide information about your NUT deployment (version, configuration, OS...) and the device, to collect logs, and to answer any follow-up questions about your situation.

Finally, note that NUT is packaged and delivered by packaging into numerous operating systems, appliances and monitoring projects, and may be bundled with third-party GUI clients. It may be wise of end-users to identify such cases and ask for help on the most-relevant forum (or several, including the NUT support channels). It is important to highlight that the NUT project releases have for a long time been essentially snapshots of better-tested code, and we do not normally issue patches to "hot-fix" any older releases.

Any improvements of NUT itself are made in the current code base, same as any other feature development, so to receive desired fixes on your system (and/or to check that they do solve your particular issue), expect to be asked to build the recent development iteration from GitHub or work with your appliance vendor to get a software upgrade.

Over time, downstream OS packaging or other integrations which use NUT, may issue patches as new package revisions, or new baseline versions of NUT, according to **their** release policies. It is not uncommon for distributions, especially "stable" flavours, to be a few years behind upstream projects.

2.3 Installing

If you are installing these programs for the first time, go read the [installation instructions](#) to find out how to do that. This document contains more information on what all of this stuff does.

2.4 Upgrading

When upgrading from an older version, always check the [upgrading notes](#) to see what may have changed. Compatibility issues and other changes will be listed there to ease the process.

2.5 Configuring and using

Once NUT is installed, refer to the [configuration notes](#) for directions.

2.6 Documentation

This is just an overview of the software. You should read the man pages, included example configuration files, and auxiliary documentation for the parts that you intend to use.

2.7 Network Information

These programs are designed to share information over the network. In the examples below, `localhost` is used as the host-name. This can also be an IP address or a fully qualified domain name. You can specify a port number if your `upsd` process runs on another port.

In the case of the program `upsc`, to view the variables on the UPS called `sparky` on the `upsd` server running on the local machine, you'd do this:

```
/usr/local/ups/bin/upsc sparky@localhost
```

The default port number is 3493. You can change this with "configure --with-port" at compile-time. To make a client talk to `upsd` on a specific port, add it after the hostname with a colon, like this:

```
/usr/local/ups/bin/upsc sparky@localhost:1234
```

This is handy when you have a mixed environment and some of the systems are on different ports.

The general form for UPS identifiers is this:

```
<upsname>[@<hostname>[:<port>]]
```

Keep this in mind when viewing the examples below.

2.8 Manifest

This package is broken down into several categories:

- **drivers** - These programs talk directly to your UPS hardware.
- **server** - `upsd` serves data from the drivers to the network.
- **clients** - They talk to `upsd` and do things with the status data.
- **cgi-bin** - Special class of clients that you can use with your web server.
- **scripts** - Contains various scripts, like the Perl and Python binding, integration bits and applications.

2.9 Drivers

These programs provide support for specific UPS models. They understand the protocols and port specifications which define status information and convert it to a form that `upsd` can understand.

To configure drivers, edit `ups.conf`. For this example, we'll have a UPS called "sparky" that uses the `apcsmart` driver and is connected to `/dev/ttyS1`. That's the second serial port on most Linux-based systems. The entry in `ups.conf` looks like this:

```
[sparky]
    driver = apcsmart
    port = /dev/ttyS1
```

To start and stop drivers, use `upsdrcvtl` or `upsdrcvctl` (installed on operating systems with a service management framework supported by NUT). By default, it will start or stop every UPS in the config file:

```
/usr/local/ups/sbin/upsdrcvtl start
/usr/local/ups/sbin/upsdrcvtl stop
```

However, you can also just start or stop one by adding its name:

```
/usr/local/ups/sbin/upsdrvctl start sparky
/usr/local/ups/sbin/upsdrvctl stop sparky
```

On operating systems with a supported service management framework, you might wrap your NUT drivers into individual services instances with:

```
/usr/local/ups/sbin/upsdrvsvct1 resync
```

and then manage those service instances with commands like:

```
/usr/local/ups/sbin/upsdrvsvct1 start sparky
/usr/local/ups/sbin/upsdrvsvct1 stop sparky
```

To find the driver name for your device, refer to the section below called "HARDWARE SUPPORT TABLE".

2.9.1 Extra Settings

Some drivers may require additional settings to properly communicate with your hardware. If it doesn't detect your UPS by default, check the driver's man page or help (-h) to see which options are available.

For example, the `usbhid-ups` driver allows you to use USB serial numbers to distinguish between units via the "serial" configuration option. To use this feature, just add another line to your `ups.conf` section for that UPS:

```
[sparky]
    driver = usbhid-ups
    port = auto
    serial = 1234567890
```

2.9.2 Hardware Compatibility List

The [Hardware Compatibility List](#) is available in the source directory (`nut-X.Y.Z/data/driver.list`), and is generally distributed with packages. For example, it is available on Debian systems as:

```
/usr/share/nut/driver.list
```

This table is also available [online](#).

If your driver has vanished, see the [FAQ](#) and [Upgrading notes](#).

2.9.3 Generic Device Drivers

NUT provides several generic drivers that support a variety of very similar models.

- The `genericups` driver supports many serial models that use the same basic principle to communicate with the computer. This is known as "contact closure", and basically involves raising or lowering signals to indicate power status.

This type of UPS tends to be cheaper, and only provides the very simplest data about power and battery status. Advanced features like battery charge readings and such require a "smart" UPS and a driver which supports it.

See the [genericups\(8\)](#) man page for more information.

- The `usbhid-ups` driver attempts to communicate with USB HID Power Device Class (PDC) UPSes. These units generally implement the same basic protocol, with minor variations in the exact set of supported attributes. This driver also applies several correction factors when the UPS firmware reports values with incorrect scale factors.

See the [usbhid-ups\(8\)](#) man page for more information.

- The `nutdrv_qx` driver supports the Megatec / Q1 protocol that is used in many brands (Blazer, Energy Sistem, Fenton Technologies, Mustek, Voltronic Power and many others).
See the [nutdrv_qx\(8\)](#) man page for more information.
- The `snmp-ups` driver handles various SNMP enabled devices, from many different manufacturers. In SNMP terms, `snmp-ups` is a manager, that monitors SNMP agents.
See the [snmp-ups\(8\)](#) man page for more information.
- The `powerman-pdu` is a bridge to the PowerMan daemon, thus handling all PowerMan supported devices. The PowerMan project supports several serial and networked PDU, along with Blade and IPMI enabled servers.
See the [powerman-pdu\(8\)](#) man page for more information.
- The `apcupsd-ups` driver is a bridge to the Apcupsd daemon, thus handling all Apcupsd supported devices. The Apcupsd project supports many serial, USB and networked APC UPS.
See the [apcupsd-ups\(8\)](#) man page for more information.

2.9.4 UPS Shutdowns

`upsdrvctl` can also shut down (power down) all of your UPS hardware.



Warning

If you play around with this command, expect your filesystems to die. Don't power off your computers unless they're ready for it:

```
/usr/local/ups/sbin/upsdrvctl shutdown
/usr/local/ups/sbin/upsdrvctl shutdown sparky
```

You should read the [Configuring automatic UPS shutdowns](#) chapter to learn more about when to use this feature. If called at the wrong time, you may cause data loss by turning off a system with a filesystem mounted read-write.

2.9.5 Power distribution unit management

NUT also provides an advanced support for power distribution units.

You should read the [NUT outlets management and PDU notes](#) chapter to learn more about when to use this feature.

2.10 Network Server

`upsd` is responsible for passing data from the drivers to the client programs via the network. It should be run immediately after `upsdrvctl` in your system's startup scripts.

`upsd` should be kept running whenever possible, as it is the only source of status information for the monitoring clients like `upsmon`.

2.11 Monitoring client

`upsmon` provides the essential feature that you expect to find in UPS monitoring software: safe shutdowns when the power fails.

In the layered scheme of NUT software, it is a client. It has this separate section in the documentation since it is so important.

You configure it by telling it about UPSes that you want to monitor in `upsmon.conf`. Each UPS can be defined as one of two possible types: a "primary" or "secondary".

2.11.1 Primary

The monitored UPS possibly supplies power to this system running `upsmon`, but more importantly — this system can manage the UPS (typically, this instance of `upsmon` runs on the same system as the `upsd` and driver(s)): it is capable and responsible for shutting it down when the battery is depleted (or in another approach, lingering to deplete it or to tell the UPS to reboot its load after too much time has elapsed and this system is still alive — meaning wall power returned at a "wrong" moment).

The shutdown of this (primary) system itself, as well as eventually an UPS shutdown, occurs after any secondary systems ordered to shut down first have disconnected, or a critical urgency threshold was passed.

If your UPS is plugged directly into a system's serial or USB port, the `upsmon` process on that system should define its relation to that UPS as a primary. It may be more complicated for higher-end UPSes with a shared network management capability (typically via SNMP) or several serial/USB ports that can be used simultaneously, and depends on what vendors and drivers implement. Setups with several competing primaries (for redundancy) are technically possible, if each one runs its own full stack of NUT, but results can be random (currently NUT does not provide a way to coordinate several entities managing the same device).

For a typical home user, there's one computer connected to one UPS. That means you would run on the same computer the whole NUT stack — a suitable driver, `upsd`, and `upsmon` in primary mode.

2.11.2 Secondary

The monitored UPS may supply power to the system running `upsmon` (or alternatively, it may be a monitoring station with zero PSUs fed by that UPS), but more importantly, this system can't manage the UPS — e.g. shut it down directly (through a locally running NUT driver).

Use this mode when you run multiple computers on the same UPS. Obviously, only one can be connected to the serial or USB port on a typical UPS, and that system is the primary. Everything else is a secondary.

For a typical home user, there's one computer connected to one UPS. That means you run a driver, `upsd`, and `upsmon` in primary mode.

2.11.3 Additional Information

More information on configuring `upsmon` can be found in these places:

- The [upsmon\(8\)](#) man page
- [Typical setups for big servers](#)
- [Configuring automatic UPS shutdowns](#) chapter
- The stock `upsmon.conf` that comes with the package

2.12 Clients

Clients talk to `upsd` over the network and do useful things with the data from the drivers. There are tools for command line access, and a few special clients which can be run through your web server as CGI programs.

For more details on specific programs, refer to their man pages.

2.12.1 upsc

`upsc` is a simple client that will display the values of variables known to `upsd` and your UPS drivers. It will list every variable by default, or just one if you specify an additional argument. This can be useful in shell scripts for monitoring something without writing your own network code.

`upsc` is a quick way to find out if your driver(s) and `upsd` are working together properly. Just run `upsc <ups>` to see what's going on, i.e.:

```
morbo:~$ upsc sparky@localhost
ambient.humidity: 035.6
ambient.humidity.alarm.maximum: NO,NO
ambient.humidity.alarm.minimum: NO,NO
ambient.temperature: 25.14
...
```

If you are interested in writing a simple client that monitors `upsd`, the source code for `upsc` is a good way to learn about using the `upsclient` functions.

See the [upsc\(8\)](#) man page and [NUT command and variable naming scheme](#) for more information.

2.12.2 upslog

`upslog` will write status information from `upsd` to a file at set intervals. You can use this to generate graphs or reports with other programs such as `gnuplot`.

2.12.3 upsrw

`upsrw` allows you to display and change the read/write variables in your UPS hardware. Not all devices or drivers implement this, so this may not have any effect on your system.

A driver that supports read/write variables will give results like this:

```
$ upsrw sparky@localhost

( many skipped )

[ups.test.interval]
Interval between self tests
Type: ENUM
Option: "1209600"
Option: "604800" SELECTED
Option: "0"

( more skipped )
```

On the other hand, one that doesn't support them won't print anything:

```
$ upsrw fenton@gearbox

( nothing )
```

`upsrw` requires administrator powers to change settings in the hardware. Refer to [upsd.users\(5\)](#) for information on defining users in `upsd`.

2.12.4 upscmd

Some UPS hardware and drivers support the notion of an instant command - a feature such as starting a battery test, or powering off the load. You can use `upscmd` to list or invoke instant commands if your hardware/drivers support them.

Use the `-l` command to list them, like this:

```
$ upscmd -l sparky@localhost
Instant commands supported on UPS [sparky@localhost]:

load.on - Turn on the load immediately
test.panel.start - Start testing the UPS panel
```

```
calibrate.start - Start run time calibration
calibrate.stop - Stop run time calibration
...
```

upscmd requires administrator powers to start instant commands. To define users and passwords in upsd, see [upsd.users\(5\)](#).

2.13 CGI Programs

The CGI programs are clients that run through your web server. They allow you to see UPS status and perform certain administrative commands from any web browser. Javascript and cookies are not required.

These programs are not installed or compiled by default. To compile and install them, first run `configure --with-cgi`, then do `make` and `make install`. If you receive errors about "gd" during configure, go get it and install it before continuing.

You can get the source here:

- <http://www.libgd.org/>

In the event that you need libpng or zlib in order to compile gd, they can be found at these URLs:

- <http://www.libpng.org/pub/png/pngcode.html>
- <http://www.zlib.net/>

2.13.1 Access Restrictions

The CGI programs use `hosts.conf` to see if they are allowed to talk to a host. This keeps malicious visitors from creating queries from your web server to random hosts on the Internet.

If you get error messages that say "Access to that host is not authorized", you're probably missing an entry in your `hosts.conf`.

2.13.2 upsstats

`upsstats` generates web pages from HTML templates, and plugs in status information in the right places. It looks like a distant relative of APC's old Powerchute interface. You can use it to monitor several systems or just focus on one.

It also can generate IMG references to `upsimage`.

2.13.3 upsimage

This is usually called by `upsstats` via IMG SRC tags to draw either the utility or outgoing voltage, battery charge percent, or load percent.

2.13.4 upsset

`upsset` provides several useful administration functions through a web interface. You can use `upsset` to kick off instant commands on your UPS hardware like running a battery test. You can also use it to change variables in your UPS that accept user-specified values.

Essentially, `upsset` provides the functions of `upsrw` and `upscmd`, but with a happy pointy-clicky interface.

`upsset` will not run until you convince it that you have secured your system. You **must** secure your CGI path so that random interlopers can't run this program remotely. See the `upsset.conf` file. Once you have secured the directory, you can enable this program in that configuration file. It is not active by default.

2.14 Version Numbering

The version numbers historically worked like this: if the middle number is odd, it's a development tree, otherwise it is the stable tree.

The past stable trees were 1.0, 1.2, 1.4, 2.0, 2.2 and 2.4, with the latest such stable tree designated 2.6. The development trees were 1.1, 1.3, 1.5, 2.1 and 2.3. Since the 2.4 release, there is no real separate development branch anymore since the code is available through a revision control system (namely, Git — or actually Subversion back then), development happens in feature branches that are eventually merged into the main trunk, and its snapshots become published releases. As a result, subsequent versions (2.7 and 2.8) were released without regard for even/odd values of the minor version component.

Since 2.7 line of releases, sources are tracked in Git revision control system, with the project ecosystem being hosted on GitHub, and any code improvements or other contributions merged through common pull request approach and custom NUT CI testing on multiple platforms.

Major release jumps are mostly due to large changes to the features list. There have also been a number of architectural changes which may not be noticeable to most users, but which can impact developers.

Since NUT v2.8.2 or so, development iterations have additional version components, to account for the amount of commits on the main branch (`master`) since the last known Git tag, and amount of commits on the developed feature branch that are unique to it compared to main branch. This allows for a reasonably growing version of stable baseline and local development, so that experimental packages can be installed as upgrades (or well-exposed downgrades).

While the NUT releases retain the semantic versioning three-component standard, interim builds (trunk snapshots and development branches) can expose a much more complex structure with the amount of commits in the trunk since last release, and amount of commits on the branch unique to it (not in the trunk). Additional data may include overall amount of commits in the current build since last release, and the git commit has identifier of the built code base.

More details can be seen in `docs/nut-versioning.adoc` file in the NUT source code base.

2.15 Backwards and Forwards Compatibility

The network protocol for the current version of NUT should be backwards-compatible all the way back to version 1.4. A newer client should fail gracefully when querying an older server.

If you need more details about cross-compatibility of older NUT releases (1.x vs. 2.x), please see the [Project history](#) chapter.

2.16 Support / Help / etc.

If you are in need of help, refer to the [Support instructions](#) in the user manual.

2.17 Hacking / Development Info

Additional documentation can be found in:

- the [Developer Guide](#),
- the [Packager Guide](#).

2.18 Acknowledgements / Contributions

The many people who have participated in creating and improving NUT are listed in the user manual [acknowledgements appendix](#).

We would like to highlight some organizations which provide continuous support to the NUT project (and many other FOSS projects) on technological and organizational sides, such as helping keep the donations transparent, NUT CI farm afloat, and public resources visible. Thanks for keeping the clocks ticking, day and night:



The "[NetworkUPSTools](#)" [organization on GitHub](#) arranges a lot of things, including source code hosting for NUT itself and several related projects, team management, projects, issue and pull request discussions, sponsorship, nut-website rendering and hosting, some automated actions, and more. . .



The [Jenkins CI](#) project and its huge plugin ecosystem provides the technological foundation for the largest island of the [self-hosted NUT CI farm](#). There is a fair amount of cross-pollination between the upstream project and community, and the development done originally for the NUT CI farm. See more at [Jenkins is the way to build multi-platform NUT](#) article.



Fosshost provided virtual machines where the multi-platform NUT CI farm with a [jenkins-dynamatrix setup](#) runs to arrange builds in numerous operating environments and a lot of toolkit versions and implementations. Some workers running on NUT community members' machines can also dial in to provide an example of their favourite platforms. Literally hundreds of NUT builds run for each iteration, to make sure NUT can always build and work everywhere. This allows us to ensure that NUT remains portable across two decades' worth of operating systems, compilers, script interpreters, tools and third-party dependencies.



The [CircleCI NUT pipeline](#) allows us to test NUT CI builds on MacOS.



The [AppVeyor NUT pipeline](#) allows us to test NUT CI builds on Windows (and publish preview tarballs with binaries).



The **DigitalOcean** droplets allow us to host NUT CI farm Jenkins controller and the build agents for multiple operating systems.



Gandi.Net took up the costs of NUT DNS hosting.



<https://opencollective.com/networkupstools> allows us to arrange monetary donations and spending, with public transparency of everything that happens.

3 Features

NUT provides many features, and is always improving. Thus this list may lag behind the current code.

Features frequently appear during the development cycles, so be sure to look at the [release notes and change logs](#) to see the latest additions.

3.1 Multiple manufacturer and device support

- Monitors many UPS, PDU, ATS, PSU and SCD models from more than 170 manufacturers with a unified interface ([Hardware Compatibility List](#)).
- Various communication types and many protocols are supported with the same common interface:
 - serial,
 - USB,
 - network (SNMP, Eaton / MGE XML/HTTP, IPMI).

3.2 Multiple architecture support

- Cross-platform — different flavors of Unix can be managed together with a common set of tools, even crossing architectures.
- This software has been reported to run on Linux distributions, the BSDs, Apple's OS X, commercial Solaris and open-source illumos distros, IRIX, HP/UX, Tru64 Unix, and AIX.
- Windows users may be able to build it directly with MSYS2, MinGW or Cygwin. There is also a port of the client-side monitoring to Windows called WinNUT.
- Your system will probably run it too. You just need a good C compiler and possibly some more packages to gain access to the serial ports. Other features, such as USB / SNMP / whatever, will also need extra software installed.

3.3 Layered and modular design with multiple processes

- Three layers: drivers, server, clients.
- Drivers run on the same host as the server, and clients communicate with the server over the network.
- This means clients can monitor any UPS anywhere as long as there is a network path between them.



Warning

Be sure to plug your network's physical hardware (switches, hubs, routers, bridges, ...) into the UPS!

3.4 Redundancy support — Hot swap/high availability power supplies

- upsmon can handle high-end servers which receive power from multiple UPSes simultaneously.
- upsmon won't initiate a shutdown until the total power situation across all source UPSes becomes critical (on battery and low battery).
- You can lose a UPS completely as long as you still have at least the minimum number of sources available. The minimum value is configurable.

3.5 Security and access control

- Manager functions are granted with per-user granularity. The admin can have full powers, while the admin's helper can only do specific non-destructive tasks such as a battery test (beware that with a worn-out battery whose replacement is a few years overdue, a "capacity/remaining runtime" test can still be destructive by powering off the load abruptly — and also such a test can cause hosts to hide into graceful shutdowns when the battery state does get critical as part of the test).
- The drivers, server, and monitoring client (upsmon) can all run as separate user IDs if this is desired for privilege separation.
- Only one tiny part of one program has root powers. upsmon starts as root and forks an unprivileged process which does the actual monitoring over the network. They remain connected over a pipe. When a shutdown is necessary, a single character is sent to the privileged process. It then calls the predefined shutdown command. In any other case, the privileged process exits. This was inspired by the auth mechanism in Solar Designer's excellent popa3d.
- The drivers and network server may be run in a chroot jail for further security benefits. This is supported directly since version 1.4 and beyond with the *chroot=* configuration directive.
- IP-based access control relies on the local firewall and [TCP Wrapper](#).
- SSL is available as a build option ("*--with-ssl*"). It encrypts sessions with upsd and can also be used to authenticate servers.

3.6 Web-based monitoring

- Comes stock with CGI-based web interface tools for UPS monitoring and management, including graphical status displays.
- Custom status web pages may be generated with the CGI programs, since they use templates to create the pages. This allows you to have status pages which fit the look and feel of the rest of your site.

3.7 Free software

- That's free beer and free speech. Licensed under the GNU General Public License version 2 or later.
- Know your systems — all source code is available for inspection, so there are no mysteries or secrets in your critical monitoring tools.

3.8 UPS management and control

- Writable variables may be edited on higher end equipment for local customization
 - Status monitoring can generate notifications (email/pager/SMS/...) on alert conditions
 - Alert notices may be dampened to only trigger after a condition persists. This avoids the usual pager meltdown when something happens and no delay is used.
 - Maintenance actions such as battery runtime calibration are available where supported by the UPS hardware.
 - Power statistics can be logged in custom formats for later retrieval and analysis
-

- All drivers are started and stopped with one common program. Starting one is as easy as starting ten: `upsdrvctl start`.
- For operating systems with a supported service management framework, you can manage the NUT drivers wrapped into independent service instances using the `upsdrcvctl` instead, and gain the benefits of automated restart as well as possibility to define further dependencies between your OS components.
- Shutdowns and other procedures may be tested without stressing actual UPS hardware by simulating status values with the dummy-ups pseudo-driver. Anything that can happen in a driver can be replicated with dummy-ups.

3.9 Monitoring diagrams

These are the most common situations for monitoring UPS hardware. Other ways are possible, but they are mostly variations of these four.

Note

These examples show serial communications for simplicity, but USB or SNMP or any other monitoring is also possible.

3.9.1 "Simple" configuration



One UPS, one computer. This is also known as "Standalone" configuration.

This is the configuration that most users will use. You need at least a driver, `upsd`, and `upsmon` running.

3.9.2 "Advanced" configuration



One UPS, multiple computers. Only one of them can actually talk to the UPS directly. That's where the network comes in:

- The Primary system runs the relevant driver, `upsd`, and `upsmon` in "primary" mode.
- The Secondary systems only run `upsmon` in "secondary" mode which all connect to `upsd` on Primary.

This is useful when you have a very large UPS that's capable of running multiple systems simultaneously. There is no longer the need to buy a bunch of individual UPSes or "sharing" hardware, since this software will handle the sharing for you.

3.9.3 "Big Box" configuration



Some systems have multiple power supplies and cords. You typically find this on high-end servers that allow hot-swap and other fun features. In this case, you run multiple drivers (one per UPS), a single `upsd`, and a single `upsmon` (as a primary for both UPS 1 and UPS 2)

This software understands that some of these servers can also run with some of the supplies gone. For this reason, every UPS is assigned a "power value" — the quantity of power supplies that it feeds on this system.

The total available "power value" is compared to the minimum that is required for that hardware. For example, if you have 3 power supplies and 3 UPSes, but only 2 supplies must be running at any given moment, the minimum would be 2.

This means that you can safely lose any one UPS and the software will handle it properly by remaining online and not causing a shut down.

3.9.4 "Bizarre" configuration



You can even have a UPS that has the serial port connected to a system that it's not feeding. Sometimes a PC will be close to a UPS that needs to be monitored, so it's drafted to supply a serial port for the purpose. This PC may in fact be getting its own power from some other UPS. This is not a problem for the set-up.

The first system ("mixed") is a Primary for UPS 1, but is only monitoring UPS 2. The other systems are Secondaries of UPS 2.

3.10 Image credits

Thanks to Eaton for providing shiny modern graphics.

3.11 Compatibility information

3.11.1 Hardware

The current list of hardware supported by NUT can be viewed [here](#).

3.11.2 Operating systems

This software has been reported to run on:

- Linux distributions,
- the BSDs,
- Apple's OS X,
- Sun Solaris and illumos,
- SGI IRIX,
- HP/UX,
- Tru64 Unix,
- AIX,
- Windows:
 - There is an older port of the client-side monitoring to Windows called WinNUT. Windows users may be able to build it directly with Cygwin.
 - Note there are also numerous third-party projects named WinNUT-Client or similar, made over decades by different enthusiasts and community members with a number of technologies underneath. If you run a program that claims such a name, locate and ask its creators for support related to the client.
 - Since NUT v2.8.1, there is an on-going effort to integrate older platform development of NUT for Windows into the main code base — allowing to run the whole stack of NUT drivers, data server and same clients as on POSIX platforms (for fancy GUI clients, see [NUT-Monitor\(8\)](#) or third-party projects). Still, as of NUT release v2.8.3, installation is complicated and there are other known imperfections (not all WIN32 code has equivalents to POSIX); for current details see NUT issues tracked on GitHub under <https://github.com/orgs/networkupstools/projects/2/views/1>

NUT is also often embedded into third-party projects like OpenWRT (or similar) based routers, NAS and other appliances, monitoring systems like Home Assistant, and provided or suggested by some UPS vendors as their software companion.

Your system will probably run it too. You just need a good C compiler and possibly some more packages to gain access to the serial ports. Other features, such as USB / SNMP / whatever, will also need extra software installed.

Success reports are welcomed to keep this list accurate.

Given its core position at the heart of your systems' lifecycle, we make it a point to have current NUT building and running anywhere, especially where older releases did work before (including "abandonware" like the servers and OSes from the turn of millennium): if those boxes are still alive and in need of power protection, they should be able to get it.

Tip

If you like how the NUT project helps protect your systems from power outages, please consider sponsoring or at least "starring" it on GitHub at <https://github.com/networkupstools/nut/> - these stars are among metrics which the larger potential sponsors consider when choosing how to help FOSS projects. Keeping the lights shining in such a large non-regression build matrix is a big undertaking!

See [acknowledgements of organizations which help with NUT CI and other daily operations](#) for an overview of the shared effort.

As a FOSS project, for over a quarter of a century we welcome contributions of both core code (drivers and other features), build recipes and other integration elements to make it work on your favourite system, documentation revisions to make it more accessible to newcomers, as well as hardware vendor cooperation with first-hand driver and protocol submissions, and just about anything else you can think of.

3.11.3 NUT Support Policy

The Network UPS Tools project is a community-made open-source effort, primarily made and maintained by people donating their spare time.

The support channels are likewise open, with preferred ones being [the NUT project issue tracker on GitHub](#) and the NUT Users mailing list, as detailed at <https://networkupstools.org/support.html> page.

Please keep in mind that any help is provided by community members just like yourself, as a best effort, and subject to their availability and experience. It is expected that you have read the Frequently Asked Questions, looked at the [NUT wiki](#), and have a good grasp about the three-layer design and programs involved in a running deployment of NUT, for a discussion to be constructive and efficient.

Be patient, polite, and prepare to learn and provide information about your NUT deployment (version, configuration, OS...) and the device, to collect logs, and to answer any follow-up questions about your situation.

Finally, note that NUT is packaged and delivered by packaging into numerous operating systems, appliances and monitoring projects, and may be bundled with third-party GUI clients. It may be wise of end-users to identify such cases and ask for help on the most-relevant forum (or several, including the NUT support channels). It is important to highlight that the NUT project releases have for a long time been essentially snapshots of better-tested code, and we do not normally issue patches to "hot-fix" any older releases.

Any improvements of NUT itself are made in the current code base, same as any other feature development, so to receive desired fixes on your system (and/or to check that they do solve your particular issue), expect to be asked to build the recent development iteration from GitHub or work with your appliance vendor to get a software upgrade.

Over time, downstream OS packaging or other integrations which use NUT, may issue patches as new package revisions, or new baseline versions of NUT, according to **their** release policies. It is not uncommon for distributions, especially "stable" flavours, to be a few years behind upstream projects.

4 Download information

This section presents the different methods to download NUT.

4.1 Source code

Note

You should always use PGP/GPG to verify the signatures before using any source code.

You can use the [following procedure](#) to do so.

4.1.1 Stable tree: 2.8

-
-
-
-
-

- [ChangeLog](#)

You can also browse the [stable source directory](#).

4.1.2 Development tree:

Code repository

The development tree is available through a Git repository hosted at [GitHub](#).

To retrieve the current development tree, use the following command:

```
:: git clone git://github.com/networkupstools/nut.git
```

OPTIONALLY you can then fetch known git tags, so semantic versions look better (based off a recent release):

```
:: cd nut
:: git fetch --tags --all
```

The `configure` script and its dependencies are not stored in Git. To generate them, ensure that `autoconf`, `automake` and `libtool` are installed, then run the following script in the directory you just checked out:

```
:: ./autogen.sh
```

Note

it is optionally recommended to have Python 2.x or 3.x, and Perl, to generate some files included into the `configure` script, presence is checked by `autotools` when it is generated. Neutered files can be just "touched" to pass the `autogen.sh` if these interpreters are not available, and effectively skip those parts of the build later on — `autogen.sh` will then advise which special environment variables to `export` in your situation and re-run it.

Then refer to the [NUT user manual](#) for more information.

Browse code

You can browse the "vanilla NUT" code at the [Main GitHub repository for NUT sources](#), and some possibly modified copies as part of packaging recipe sources of operating system distributions, as listed below.

Snapshots

GitHub has several download links for repository snapshots (for particular tags or branches), but you will need a number of tools such as `autoconf`, `automake` and `libtool` to use these snapshots to generate the `configure` script and some other files.

After you `configure` the source workspace, a `make dist-hash` recipe would create the snapshot tarballs which do not require the `auto*` tools, and their checksum files, such as those available on the NUT website and attached to [GitHub Releases page](#).

4.1.3 Older versions

[Browse source directory](#)

4.2 Binary packages

Note

The only official releases from this project are source code.

NUT is already available in the following operating systems (and [likely more](#)):

- [Repology report on NUT](#) lists 745 entries about NUT, as of this writing
 - Linux:
 - [42ITy.org](#) packaging recipes for Debian-based releases
 - [Debian Salsa recipes](#) and [Debian packages](#)
 - [Ubuntu packages](#)
 - [Fedora Rawhide recipes](#) and [Red Hat / Fedora packages](#)
 - [Arch Linux recipe](#) and [Arch Linux package info](#)
 - [Gentoo Linux recipe](#) and [Gentoo Linux package info](#)
 - [Novell SUSE / openSUSE official package base recipe](#) and [Novell SUSE / openSUSE official package development recipe](#), and [Novell SUSE / openSUSE official package overview](#)
 - Numerous other recipes on Open Build System (not only by SUSE)
 - [OpenWRT recipes](#)
 - [Slackware package overview](#)
 - [Void Linux recipes](#)
 - BSD systems:
 - [FreeBSD package recipe \(devel\)](#), [FreeBSD package recipe](#) and [FreeBSD package overview](#)
 - [NetBSD recipe](#) and [NetBSD package overview](#)
 - [OpenBSD recipe](#)
 - [FreeNAS iocage-ports recipe](#), [FreeNAS 9.3 docs on UPS integration](#) and [FreeNAS 11.3-U5 docs on UPS integration](#)
 - Mac OS X:
 - [Fink recipe](#) and [Fink package overview](#)
 - [MacPorts recipe](#)
 - illumos/Solaris:
 - [OpenIndiana oi-userland recipe](#) and [OpenIndiana latest rolling builds](#)
 - Windows (complete port, Beta):
 - Current regular CI builds are available as tarballs with binaries from [Appveyor CI](#)—but it may be difficult to locate specifically the master-branch builds. See [NUT for Windows wiki article](#) for these details, and more.
 - [\(OBSOLETE\) Windows MSI installer 2.6.5-6](#)
-

4.3 Java packages

- The jNut package has been split into its own [GitHub repository](#).
- NUT Java support (client side, Beta) [jNUT 0.2-SNAPSHOT](#)
- NUT Java Web support (client side using REST, Beta) [jNutWebAPI 0.2-SNAPSHOT \(sources\)](#)

4.4 Virtualization packages

4.4.1 VMware

- NUT client for VMware ESXi (several versions of both; offsite, by René Garcia). Since the hypervisor manager environment lacks access to hardware ports, this package only includes the `upsmon` client integration, and a NUT server must run in a VM with passed-through ports.

See [NUT and VMware \(ESXi\) page on NUT Wiki](#) for more community-contributed details.

Note that the VIB package versioning is independent of NUT or VMware versions, they are however mentioned in downloadable file names. As of this writing, there are builds spanning VMware ESXi 5.0-8.0 and NUT 2.7.4-2.8.0.



Warning

This module is provided "as is" and is not approved by VMware, you may lose VMware support if you install it. Use it at your own risks.

- [GitHub repository with build recipes](#), including [binary releases](#)
- [Original blog entry \(French\)](#)
- [Historic details of the recipe evolution](#)
- [VIB package \(in fact automatically redirects to latest build\)](#)

5 Installation instructions

This chapter describes the various methods for installing Network UPS Tools.

Whenever it is possible, prefer [installing from packages](#). Packagers have done an excellent and hard work at improving NUT integration into their operating system. On the other hand, distributions and appliances tend to package "official releases" of projects such as NUT, and so do not deliver latest and greatest fixes, new drivers, bugs and other features.

Stable distributions also tend to deliver minor fixes to the version of third-party software (like NUT) a particular release of the operating system has initially delivered, so those release lines can be years behind development in terms of new features (or bug fixes, if they are not trivial to patch into the old code base snapshot used to create the package).

5.1 Installing from source

These are the essential steps for compiling and installing this software from distribution archives (usually "release tarballs") which include a pre-built copy of the `configure` script and some other generated source files.

To build NUT from a Git checkout you may need some additional tools (referenced just a bit below) and run `./autogen.sh` to generate the needed files. For common developer iterations, porting to new platforms, or in-place testing, running the `./ci_build.sh` script can be helpful. The "[Building NUT for in-place upgrades or non-disruptive tests](#)" section details some more hints about such workflow, including some `systemd` integration.

The NUT [Packager Guide](#), which presents the best practices for installing and integrating NUT, is also a good reading.

The [NUT Quality Assurance Guide](#) (or `docs/config-prereqs.txt` in NUT sources for up-to-date information) document suggests prerequisite packages with tools and dependencies available and needed to build and test as much as possible of NUT on numerous platforms, written from perspective of CI testing (if you are interested in getting updated drivers for a particular device, you might select a sub-set of those suggestions).

Note

This "Config Prereqs" document for latest NUT iteration can be found at <https://github.com/networkupstools/nut/blob/master/docs/config-prereqs.txt> or as `docs/config-prereqs.txt` in your build workspace (from Git or tarball).

Keep in mind that...

- the paths shown below are the default values you get by just calling `configure` by itself. If you have used `--prefix` or similar, things will be different. Also, if you didn't install this program from source yourself, the paths will probably have a number of differences.
- by default, your system probably won't find the man pages, since they install to `/usr/local/ups/man`. You can fix this by editing your `MANPATH`, or just do this:

```
man -M /usr/local/ups/man <man page>
```

- if your favorite system offers up to date binary packages, you should always prefer these over a source installation (unless there are known deficiencies in the package or one is too obsolete). Along with the known advantages of such systems for installation, upgrade and removal, there are many integration issues that have been addressed.
-

5.1.1 Prepare your system

System User creation

Create at least one system user and a group for running this software. You might call them "ups" and "nut". The exact names aren't important as long as you are consistent.

The process for doing this varies from one system to the next, and explaining how to add users is beyond the scope of this document.

For the purposes of this document, the user name and group name will be *ups* and *nut* respectively.

Be sure the new user is a member of the new group! If you forget to do this, you will have problems later on when you try to start `upsd`.

5.1.2 Build and install

Note

See also [Building NUT for in-place upgrades or non-disruptive tests](#).

Configuration

Configure the source tree for your system. Add the `--with-user` and `--with-group` switch to set the user name and group that you created above.

```
./configure --with-user=ups --with-group=nut
```

If you need any other switches for `configure`, add them here. For example:

- to build and install USB drivers, add `--with-usb` (note that you need to install `libusb` development package or files).
 - to build and install SNMP drivers, add `--with-snmp` (note that you need to install `libsnmp` development package or files).
 - to build and install CGI scripts, add `--with-cgi`.
-

See [Configure options](#) from the User Manual, docs/configure.txt or `./configure --help` for all the available options.

If you alter paths with additional switches, be sure to use those new paths while reading the rest of the steps.

Reference: [Configure options](#) from the User Manual.

Build the programs

```
make
```

This will build the NUT client and server programs and the selected drivers. It will also build any other features that were selected during [configuration](#) step above.

Note

NUT is regularly tested with GNU, BSD and Sun implementations of `make`.

Installation

Note

You should now gain privileges for installing software if necessary, e.g.:

```
su -
```

or prefix installation (not build/check!) commands with `sudo`, e.g.:

```
sudo make install ...
```

Note

If you install NUT for direct consumption on the system that has just built it from source, you might be aided for some of the steps listed and explained below by running (as `root`):

```
make install-as-root
```

This should not only install the software, but also create directories such as the state path, assign ownership and access permissions to certain directories and sample configuration files which NUT user and/or group accounts should be able to read and/or write, and, on some platforms, also (re-)start the NUT daemons/services.

See above "System User creation", that should be done first!

Install the files to a system level directory:

```
make install
```

This will install the compiled programs and man pages, as well as some data files required by NUT. Any optional features selected during configuration will also be installed.

This will also install sample versions of the NUT configuration files. Sample files are installed with names like `ups.conf.sample` so they will not overwrite any existing real config files you may have created.

If you are packaging this software, then you will probably want to use the `DESTDIR` variable to redirect the build into another place, also known as a "prototype directory" or a "staging area", i.e.:

```
make DESTDIR=/tmp/package install
make DESTDIR=/tmp/package install-conf
```

State path creation

Note

See above about `make install-as-root`, if you use that — skip this step here.

Create the state path directory for the driver(s) and server to use for storing UPS status data and other auxiliary files, and make it group-writable by the group of the system user you created, e.g.:

```
mkdir -p /var/state/ups
chmod 0770 /var/state/ups
chown root:nut /var/state/ups
```

Ownership and permissions

Set ownership data and permissions on your serial or USB ports that go to your UPS hardware. Be sure to limit access to just the user you created earlier.

These examples assume the second serial port (ttyS1) on a typical Slackware system. On FreeBSD, that would be cuaa1. Serial ports vary greatly, so yours may be called something else.

```
chmod 0660 /dev/ttyS1
chown root:nut /dev/ttyS1
```

The setup for USB ports is slightly more complicated. Device files for USB devices, such as `/proc/bus/usb/002/001`, are usually created "on the fly" when a device is plugged in, and disappear when the device is disconnected. Moreover, the names of these device files can change randomly. To set up the correct permissions for the USB device, you may need to set up (operating system dependent) hotplugging scripts. Sample scripts and information are provided in the `scripts/hotplug` and `scripts/udev` directories. For most users, the hotplugging scripts will be installed automatically by "make install".

(If you want to try if a driver works without setting up hotplugging, you can add the "-u root" option to `upsd`, `upsmon`, and `drivers`; this should allow you to follow the below instructions. However, don't forget to set up the correct permissions later!).

Note

If you are using something like `udev` or `devd`, make sure these permissions stay set across a reboot. If they revert to the old values, your drivers may fail to start.

You are now ready to configure NUT, and start testing and using it.

You can jump directly to the [NUT configuration](#).

5.2 Building NUT for in-place upgrades or non-disruptive tests

Note

The NUT GitHub Wiki article at <https://github.com/networkupstools/nut/wiki/Building-NUT-for-in-place-upgrades-or-non-disruptive-tests> may contain some more hints as contributed by the community.

5.2.1 Overview

Since late 2022/early 2023 NUT codebase supports "in-place" builds which try their best to discover the configuration of an earlier build (configuration and run-time paths and OS accounts involved, maybe an exact configuration if stored in deployed binaries).

This optional mode is primarily intended for several use-cases:

- Test recent GitHub "master" branch or a proposed PR to see if it solves a practical problem for a particular user;
- Replace an existing deployment, e.g. if OS-provided packages deliver obsolete code, to use newer NUT locally in "production mode".
 - In such cases ideally get your distribution, NAS vendor, etc. to provide current NUT — and benefit from a better integrated and tested product.

Note that "just testing" often involves building the codebase and new drivers or tools in question, and running them right from the build workspace (without installing into the system and so risking an unpredictable-stability state). In case of testing new driver builds, note that you would need to stop the normally running instances to free up the communications resources (USB/serial ports, etc.), run the new driver program in data-dump mode, and restart the normal systems operations.

Such tests still benefit from matching the build configuration to what is already deployed, in order to request same configuration files and system access permissions (e.g. to own device nodes for physical-media ports involved, and to read the production configuration files).

Pre-requisites

The [NUT Quality Assurance Guide](#) (or `docs/config-prereqs.txt` in NUT sources for up-to-date information) document details tools and dependencies that were added on NUT CI build environments, which now cover many operating systems. This should provide a decent starting point for the build on yours (PRs to update the document are welcome!)

Note that unlike distribution tarballs, Git sources do not include a `configure` script and some other files — these should be generated by running `autogen.sh` (or `ci_build.sh` that calls it).

Getting the right sources

To build the current tip of development iterations (usually after PR merges that passed CI, reviews and/or other tests), just clone the NUT repository and "master" branch should get checked out by default (also can request that explicitly, per example posted below).

If you want to quickly test a particular pull request, see the link on top of the PR page that says ... wants to merge ... from : ... and copy the proposed-source URL of that "from" part.

For example, in some PR this says `jimklimov:issue-1234` and links to `https://github.com/jimklimov/nut/tree/i`. For manual git-cloning, just paste that URL into the shell and replace the `/tree/` with `"-b"` CLI option for branch selection; it also helps to keep the workspace directory name dedicated to that PR, like this:

```
;; cd /tmp
### Checkout https://github.com/jimklimov/nut/tree/issue-1234
;; git clone https://github.com/jimklimov/nut -b issue-1234 nut-issue-1234
;; cd nut-issue-1234
### OPTIONALLY fetch known git tags, so semantic versions look better
;; git fetch --tags --all
### Proceed with build (common instructions below)
```

5.2.2 Testing with CI helper

Note

This uses the `ci_build.sh` script to arrange some rituals and settings, in this case primarily to default the choice of drivers to auto-detection of what can be built, and to skip building documentation. Also note that this script supports many other scenarios for CI and developers, managed by `BUILD_TYPE` and other environment variables, which are not explored here.

An "in-place" *testing* build and run would probably go along these lines:

```

;; cd /tmp
;; git clone -b master https://github.com/networkupstools/nut
;; cd nut
### OPTIONALLY fetch known git tags, so semantic versions look better
;; git fetch --tags --all
### Proceed with build
;; ./ci_build.sh inplace
### Temporarily stop your original drivers
;; ./drivers/nutdrv_qx -a DEVNAME_FROM_UPS_CONF -d1 -DDDDDD \
    # -x override...=... -x subdriver=...
### Can start back your original drivers
### Analyze and/or post back the data-dump

```

Note

To probe a device for which you do not have an `ups.conf` section yet, you must specify `-s name` and all config options (including port) on command-line with `-x` arguments, e.g.:

```

;; ./drivers/nutdrv_qx -s temp-ups \
    -d1 -DDDDDD -x port=auto \
    -x vendorid=... -x productid=... \
    -x subdriver=...

```

5.2.3 Replacing a NUT deployment

While `ci_build.sh inplace` can be a viable option for preparation of local builds, you may want to have precise control over configure options (e.g. choice of required drivers, or enabled documentation).

A sound starting point would be to track down packaging recipes used by your distribution (e.g. [RPM spec](#) or [DEB rules](#) files, etc.) to detail the same paths if you intend to replace those, and copy the parameters for `configure` script from there—especially if your system is not currently running NUT v2.8.1 or newer (which embeds this information to facilitate in-place upgrade rebuilds).

Note that the primary focus of in-place automated configuration mode is about critical run-time options, such as OS user accounts, configuration location and state/PID paths, so it alone might not replace your driver binaries that the package would put into an obscure location like `/lib/nut`. It would however install init-scripts or systemd units that would refer to new locations specified by the current build, so such old binaries would just consume disk space but not run.

It is recommended to first try installing into a prototyping area, so you can review which files get delivered and perhaps which locations you may have to tune for a more tightly tailored replacement of an older (packaged) installation, in case whole swathes of files that you expect to be present in the current system (libraries, drivers, docs) are not getting installed by the new build into same path names:

```

;; rm -rf /tmp/nut ; make DESTDIR=/tmp/nut install -j 8
;; (cd /tmp/nut && find . | sort) | while read N ; \
    do [ -e "$N" ] || echo "=== MISSING: $N" >&2 ; done

```

Replacing any NUT deployment

Note

For deployments on OSes with `systemd` see the next section.

This goes similar to usual build and install from Git:

```

;; cd /tmp
;; git clone https://github.com/networkupstools/nut
;; cd nut
### OPTIONALLY fetch known git tags, so semantic versions look better
;; git fetch --tags --all
### Proceed with build
;; ./autogen.sh
;; ./configure --enable-inplace-runtime # --maybe-some-other-options
;; make -j 4 all && make -j 4 check && sudo make install

```

Note that `make install` does not currently handle all the nuances that packaging installation scripts would, such as customizing filesystem object ownership, daemon restarts, etc. or even creating locations like `/var/state/ups` and `/var/run/nut` as part of the `make` target (but e.g. the delivered `systemd-tmpfiles` configuration can handle that for a large part of the audience). This aspect is tracked as [issue #1298](#)

At this point you should revise the locations for PID files (e.g. `/var/run/nut`) and pipe files (e.g. `/var/state/ups`) that they exist and permissions remain suitable for NUT run-time user selected by your configuration, and typically stop your original NUT drivers, data-server (`upsd`) and `upsmon`, and restart them using the new binaries.

Replacing a systemd-enabled NUT deployment

For modern Linux distributions with `systemd` this replacement procedure could be enhanced like below, to also re-enable services (creating proper symlinks) and to get them started:

```

;; cd /tmp
;; git clone https://github.com/networkupstools/nut
;; cd nut
### OPTIONALLY fetch known git tags, so semantic versions look better
;; git fetch --tags --all
### Proceed with build
;; ./autogen.sh
;; ./configure --enable-inplace-runtime # --maybe-some-other-options
;; make -j 4 all && make -j 4 check && \
{ sudo systemctl stop nut-monitor nut-server || true ; } && \
{ sudo systemctl stop nut-driver.service || true ; } && \
{ sudo systemctl stop nut-driver.target || true ; } && \
{ sudo systemctl stop nut.target || true ; } && \
sudo make install && \
sudo systemctl daemon-reload && \
sudo systemd-tmpfiles --create && \
sudo systemctl disable nut.target nut-driver.target \
    nut-monitor nut-server nut-driver-enumerator.path \
    nut-driver-enumerator.service && \
sudo systemctl enable nut.target nut-driver.target \
    nut-monitor nut-server nut-driver-enumerator.path \
    nut-driver-enumerator.service && \
{ sudo systemctl restart udev || true ; } && \
sudo systemctl restart nut-driver-enumerator.service \
    nut-monitor nut-server

```

Note the several attempts to stop old service units—naming did change from 2.7.4 and older releases, through 2.8.0, and up to current codebase. Most of the NUT units are now `WantedBy=nut.target` (which is in turn `WantedBy=multi-user.target` and so bound to system startup). You should only `systemctl enable` those units you need on this system—this allows it to not start the daemons you do not need (e.g. not run `upsd` NUT data server on systems which are only `upsmon` secondary clients).

The `nut-driver-enumerator` units (and corresponding shell script) are part of a new feature introduced in NUT 2.8.0, which automatically discovers `ups.conf` sections and changes to their contents, and manages instances of a `nut-driver@.service` definition.

You may also have to restart (or reload if supported) some system services if your updates impact them, like `udev` for updates USB support (note also [PR #1342](#) regarding the change from `udev.rules` to `udev.hwdb` file with NUT v2.8.0 or later—you may have to remove the older file manually).

Iterating with a systemd deployment

If you are regularly building NUT from GitHub "master" branch, or iterating local development branches of your own, you **may** get away with shorter constructs to just restart the services after installing newly built files (if you know there were no changes to unit file definitions and dependencies), e.g.:

```
;; cd /tmp
;; git clone https://github.com/networkupstools/nut
;; cd nut
;; git checkout -b issue-1234 ### your PR branch name, arbitrary
### OPTIONALLY fetch known git tags, so semantic versions look better
;; git fetch --tags --all
### Proceed with build
;; ./autogen.sh
;; ./configure --enable-inplace-runtime # --maybe-some-other-options
### Iterate your code changes (e.g. PR draft), build and install with:
;; make -j 4 all && make -j 4 check && \
    sudo make install && \
    sudo systemctl daemon-reload && \
    sudo systemd-tmpfiles --create && \
    sudo systemctl restart \
        nut-driver-enumerator.service nut-monitor nut-server
```

Note that to contribute your work back to upstream NUT codebase, you would need to create a "fork" of <https://github.com/networkupstools/nut> on GitHub, then `git remote add USERNAME https://github.com/USERNAME/nut`, maybe refresh the workspace index with `git fetch --all`, and finally `git push USERNAME` (possibly follow further instructions from `git` tooling) to create the pull request. For more details, see `docs/developers.txt` in NUT sources.

5.2.4 Next steps after an in-place upgrade

You can jump directly to the [NUT configuration](#) if you need to revise the settings for your new NUT version, take advantage of new configuration options, etc.

Check the `NEWS.adoc` and `UPGRADING.adoc` files in your checked-out Git workspace to review features that should be present in your new build.

5.3 Installing from packages

This chapter describes the specific installation steps when using binary packages that exist on various major systems.

5.3.1 Debian, Ubuntu and other derivatives

Note

NUT is packaged and well maintained in these systems. The official Debian packager used to be part of the NUT Team.

Using your preferred method (`apt-get`, `aptitude`, `Synaptic`, ...), install the `nut` package, and optionally the following:

- *nut-cgi*, if you need the CGI (HTML) option,
- *nut-snmp*, if you need the snmp-ups driver,
- *nut-xml*, for the netxml-ups driver,
- *nut-powerman-pdu*, to control the PowerMan daemon (PDU management)
- *nut-dev*, if you need the development files.

Configuration files are located in `/etc/nut`. `nut.conf(5)` must be edited to be able to invoke `/etc/init.d/nut`

Note

Ubuntu users can access the APT URL installation by clicking on [this link](#).

5.3.2 Mandriva

Note

NUT is packaged and well maintained in these systems. The official Mandriva packager is part of the NUT Team.

Using your preferred method (`urpmi`, `RPMdrake`, ...), install one of the two below packages:

- *nut-server* if you have a *standalone* or *netserver* installation,
- *nut* if you have a *netclient* installation.

Optionally, you can also install the following:

- *nut-cgi*, if you need the CGI (HTML) option,
- *nut-devel*, if you need the development files.

5.3.3 SUSE / openSUSE

Note

NUT is packaged and well maintained in these systems. The official SUSE packager is part of the NUT Team.

Install the *nut-classic* package, and optionally the following:

- *nut-drivers-net*, if you need the snmp-ups or the netxml-ups drivers,
- *nut-cgi*, if you need the CGI (HTML) option,
- *nut-devel*, if you need the development files,

Note

SUSE and openSUSE users can use the [one-click install method](#) to install NUT.

5.3.4 Red Hat, Fedora and CentOS

Note

NUT is packaged and well maintained in these systems. The official Red Hat packager is part of the NUT Team.

Using your preferred method (yum, Add/Remove Software, ...), install one of the two below packages:

- *nut* if you have a *standalone* or *netserver* installation,
- *nut-client* if you have a *netclient* installation.

Optionally, you can also install the following:

- *nut-cgi*, if you need the CGI (HTML) option,
- *nut-xml*, if you need the netxml-ups driver,
- *nut-devel*, if you need the development files.

5.3.5 FreeBSD

You can either install NUT as a binary package or as a port.

Binary package

To install NUT as a package execute:

```
# pkg install nut
```

Port

The port is located under `sysutils/nut`. Use `make config` to select configuration options, e.g. to build the optional CGI scripts. To install it, use:

```
# make install clean
```

USB UPS on FreeBSD

For USB UPS devices the NUT package/port installs `devd` rules in `/usr/local/etc/devd/nut-usb.conf` to set USB device permissions. *devd* needs to be restarted for these rules to apply:

```
# service devd restart
```

(Re-)connect the device after restarting *devd* and check that the USB device has the proper permissions. Check the last entries of the system message buffer. You should find an entry like:

```
# dmesg | tail
[...]
ugen0.2: <INNO TECH USB to Serial> at usb0
```

The device file must be owned by group `uucp` and must be group read-/writable. In the example from above this would be

```
# ls -l /dev/ugen0.2
crw-rw----  1 root  uucp  0xa5 Mar 12 10:33 /dev/ugen0.2
```

If the permissions are not correct, verify that your device is registered in `/usr/local/etc/devd/nut-usb.conf`. The vendor and product id can be found using:

```
# usbconfig -u 0 -a 2 dump_device_desc
```

where `-u` specifies the USB bus number and `-a` specifies the USB device index.

5.3.6 Windows

Windows binary package

Note

NUT binary package built for Windows platform was last issued for a much older codebase (using NUT v2.6.5 as a baseline). While the current state of the codebase you are looking at aims to refresh the effort of delivering NUT on Windows, the aim at the moment is to help developers build and modernize it after a decade of blissful slumber, and packages are not being regularly produced yet. Functionality of such builds varies a lot depending on build environment used. This effort is generally tracked at <https://github.com/orgs/networkupstools/projects/2/views/1> and help would be welcome!

It should currently be possible to build the codebase in native Windows with MSYS2/MinGW and cross-building from Linux with mingw (preferably in a Debian/Ubuntu container). Refer to [Prerequisites for building NUT on different OSES](#) and [scripts/Windows/README.adoc](#) file for respective build environment preparation instructions.

Note that to use NUT for Windows, non-system dependency DLL files must be located in same directory as each EXE file that uses them. This can be accomplished for FOSS libraries (copying them from the build environment) by calling `make install-win-bundle DESTDIR=/some/valid/location` easily.

Archives with binaries built by recent iterations of continuous integration jobs should be available for exploration on the respective CI platforms.

Information below may be currently obsolete, but the NUT project wishes it to become actual and factual again :)

NUT binary package built for Windows platform comes in a `.msi` file.

If you are using Windows 95, 98 or Me, you should install [Windows Installer 2.0](#) from Microsoft site.

If you are using Windows 2000 or NT 4.0, you can [download it here](#).

Newer Windows releases should include the Windows Installer natively.

Run `NUT-Installer.msi` and follow the wizard indications.

If you plan to use an UPS which is locally connected to an USB port, you have to install [libUSB-win32](#) on your system. Then you must install your device via libusb's "Inf Wizard".

Note

If you intend to build from source, relevant sources may be available at <https://github.com/mcuae/libusb-win32> and keep in mind that it is a variant of libusb-0.1. Current NUT supports libusb-1.0 as well, and that project should have Windows support out of the box (but it was not explored for NUT yet).

If you have selected default directory, all configuration files are located in `C:\Program Files\NUT\ups\etc`

Building for Windows

For suggestions about setting up the NUT build environment variants for Windows, please see link:docs/config-prereqs.txt and/or link:scripts/Windows/README.adoc files. Note this is rather experimental at this point.

If using USB drivers, such as `usbhid-ups`, `nutdrv_qx` or `blazer_usb`, you may benefit from the [Zadig](#) tool to pin the low-level drivers needed for LibUSB and further NUT to interact with the hardware. Otherwise, OS-provided (e.g. "HID Battery") or third-party drivers may attach and "steal" bytes from the data stream needed by LibUSB rendering it invalid.

Note

For more details, see [NUT issue #1690](#) and the [NUT for Windows](#) article on NUT GitHub Wiki.

5.3.7 Runtime configuration

You are now ready to configure NUT, and start testing and using it.

You can jump directly to the [NUT configuration](#).

6 Configuration notes

This chapter describes most of the configuration and use aspects of NUT, including establishing communication with the device and configuring safe shutdowns when the UPS battery runs out of power.

There are many programs and [features](#) in this package. You should check out the [NUT Overview](#) and other accompanying documentation to see how it all works.

Note

NUT does not currently provide proper graphical configuration tools. However, there is now support for [Augeas](#), which will enable the easier creation of configuration tools.

The [nutconf\(8\)](#) tool should also help with programmatic manipulation of various NUT configuration files.

Moreover, [nut-scanner\(8\)](#) is available to discover supported devices (USB, SNMP, Eaton XML/HTTP and IPMI) and NUT servers (using Avahi or the classic connection method).

6.1 Details about the configuration files

6.1.1 Generalities

All configuration files within this package are parsed with a common state machine, which means they all can use a number of extras described here.

First, most of the programs use an upper-case word to declare a configuration directive. This may be something like MONITOR, NOTIFYCMD, or ACCESS. The case does matter here. "monitor" won't be recognized.

Next, the parser does not care about whitespace between words. If you like to indent things with tabs or spaces, feel free to do it here.

If you need to set a value to something containing spaces, it has to be contained within "quotes" to keep the parser from splitting up the line. That is, you want to use something like this:

```
SHUTDOWNCMD "/sbin/shutdown -h +0"
```

Without the quotes, it would only see the first word on the line.

OK, so let's say you really need to embed that kind of quote within your configuration directive for some reason. You can do that too.

```
NOTIFYCMD "/bin/notifyme -foo -bar \"hi there\" -baz"
```

In other words, \ can be used to escape the ".

Finally, for the situation where you need to put the \ character into your string, you just escape it.

```
NOTIFYCMD "/bin/notifyme c:\\dos\\style\\path"
```

The \ can actually be used to escape any character, but you only really need it for \, ", and # as they have special meanings to the parser.

When using file names with space characters, you may end up having tricky things since you need to write them inside "" which must be escaped:

```
NOTIFYCMD "\"c:\\path with space\\notifyme\" \"c:\\path with space\\name\""
```

is the comment character. Anything after an unescaped # is ignored.

Something like this...

```
identity = my#lups
```

will actually turn into `identity = my`, since the # stops the parsing. If you really need to have a # in your configuration, then escape it.

```
identity = my\\#lups
```

Much better.

The = character should be used with care too. There should be only one "simple" = character in a line: between the parameter name and its value. All other = characters should be either escaped or within "quotes".

```
password = 123=123
```

is incorrect. You should use:

```
password = 123\\=123
```

or:

```
password = "123=123"
```

6.1.2 Line spanning

You can put a backslash at the end of the line to join it to the next one. This creates one virtual line that is composed of more than one physical line.

Also, if you leave the "" quote container open before a newline, it will keep scanning until it reaches another one. If you see bizarre behavior in your configuration files, check for an unintentional instance of quotes spanning multiple lines.

6.2 Basic configuration

This chapter describes the base configuration to establish communication with the device.

This will be sufficient for PDU. But for UPS and SCD, you will also need to configure [automatic shutdowns for low battery events](#).



On operating systems with service management frameworks (such as Linux `systemd` and Solaris/illumos SMF), the life-cycle of driver, data server and monitoring client daemons is managed respectively by `nut-driver` (multi-instance service), `nut-server` and `nut-monitor` services. These are in turn wrapped by an "umbrella" service (or `systemd` "target") conveniently called `nut` which allows to easily start or stop all those of the bundled services, which are enabled on a particular deployment.

6.2.1 Driver configuration

Create one section per UPS in `ups.conf(5)` file.

Note

The default path for a source installation is `/usr/local/ups/etc`, while packaged installation will vary. For example, `/etc/nut` is used on Debian and derivatives, while `/etc/ups` or `/etc/upsd` is used on RedHat and derivatives.

To find out which driver to use, check the [Hardware Compatibility List](#), or `data/driver.list(.in)` source file.

Once you have picked a driver, create a section for your UPS in `ups.conf`. You must supply values at least for "driver" and "port".

Some drivers may require other flags or settings. The "desc" value is optional, but is recommended to provide a better description of what useful load your UPS is feeding.

A typical device without any extra settings looks like this:

```
[mydevice]
    driver = mydriver
    port = /dev/ttyS1
    desc = "Workstation"
```

Note

USB drivers (such as `usbhid-ups` for non-SHUT mode, `nutdrv_qx` for non-serial mode, `bcmxcp_usb`, `tripplite_usb`, `blazer_usb`, `riello_usb` and `richcomm_usb`) are special cases and ignore the *port* value.

You must still set this value, but it does not matter what you set it to; a common and good practice is to set *port* to `auto`, but you can put whatever you like.

If you only own one USB UPS, the driver will find it automatically if it matches the identifiers are built into that driver.

If you own more than one, refer to the driver's manual page for more information on matching a specific device, or trying specific subdriver or protocol options with a currently unknown device.

Note

On Windows systems, the second serial port (COM2), equivalent to `/dev/ttyS1` on Linux, would be `\\\\.\\COM2`.

References: [ups.conf\(5\)](#), [nutupsdrv\(8\)](#), [bcmxcp_usb\(8\)](#), [blazer_usb\(8\)](#), [nutdrv_qx\(8\)](#), [richcomm_usb\(8\)](#), [riello_usb\(8\)](#), [tripplite_usb\(8\)](#), [usbhid-ups\(8\)](#)

6.2.2 Starting the driver(s) in legacy operating systems

Generally, you can just start the driver(s) for your hardware (all sections defined in `ups.conf`) using the following command:

```
;; upsdrvctl start
```

Make sure the driver doesn't report any errors. It should show a few details about the hardware and then enter the background. You should get back to the command prompt a few seconds later. For reference, a successful start of the `usbhid-ups` driver looks like this:

```
# upsdrvctl start
Network UPS Tools - Generic HID driver 0.34 (2.4.1)
USB communication driver 0.31
Using subdriver: MGE HID 1.12
Detected EATON - Ellipse MAX 1100 [ADKK22008]
```

If the driver doesn't start cleanly, make sure you have picked the right one for your hardware. You might need to try other drivers by changing the "driver=" value in *ups.conf*.

Be sure to check the driver's man page to see if it needs any extra settings in *ups.conf* to detect your hardware.

If it says `can't bind /var/state/ups/...` or similar, then your state path probably isn't writable by the driver. Check the [permissions and mode on that directory](#) vs. the user account your driver starts as.

After making changes, try the [Ownership and permissions](#) step again.

6.2.3 Driver(s) as a service

On operating systems with init-scripts managing life-cycle of the operating environment, the `upsdrvctl` program is also commonly used in those scripts. It has a few downsides, such as that if the device was not accessible during OS startup and the driver connection timed out, it would remain not-started until an administrator (or some other script) "kicks" the driver to retry startup. Also, startup of the `upsd` data server daemon and its clients like `upsmon` is delayed until all the NUT drivers complete their startup (or time out trying).

This can be a big issue on systems which monitor multiple devices, such as big servers with multiple power sources, or administrative workstations which monitor a datacenter full of UPSes.

For this reason, NUT starting with version 2.8.0 supports startup of its drivers as independent instances of a `nut-driver` service under the Linux `systemd` and Solaris/illumos SMF service-management frameworks (corresponding files and scripts may be not pre-installed in packaging for other systems).

Such service instances have their own and independent life-cycle, including parallel driver start and stop processing, and retries of startup in case of failure as implemented by the service framework in the OS. The Linux `systemd` solution also includes a `nut-driver.target` as a checkpoint that all defined drivers have indeed started up (as well as being a singular way to enable or disable startup of drivers).

In both cases, a service named `nut-driver-enumerator` is registered, and when it is (re-)started it scans the currently defined device sections in *ups.conf* and the currently defined instances of `nut-driver` service, and brings them in sync (adding or removing service instances), and if there were changes — it restarts the corresponding drivers (via service instances) as well as the data server which only reads the list of sections at its startup. This helper service should be triggered whenever your system (re-)starts the `nut-server` service, so that it runs against an up-to-date list of NUT driver processes.

Two service bundles are provided for this feature: a set of `nut-driver-enumerator-daemon*` units starts the script as a daemon to regularly inspect and apply the NUT configuration to OS service unit wrappings (mainly intended for monitoring systems with a dynamic set of monitored power devices, or for systems where filesystem events monitoring is not a clockwork-reliable mechanism to 100% rely on); while the other `nut-driver-enumerator.*` units run the script once per triggering of the service (usually during boot-up; configuration file changes can be detected and propagated by `systemd` most of the time, but not by SMF out of the box).

A service-oriented solution also allows to consider that different drivers have different dependencies — such as that networked drivers should begin startup after IP addresses have been assigned, while directly-connected devices might need nothing beside a mounted filesystem (or an activated USB stack service or device rule, in case of Linux). Likewise, systems administrators can define further local dependencies between services and their instances as needed on particular deployments.

This solution also adds the `upsdrvsvctl` script to manage NUT drivers as system service instances, whose CLI mimics that of `upsdrvctl` program. One addition is the `resync` argument to trigger `nut-driver-enumerator`, another is a `list` argument to display current mappings of service instances to NUT driver sections. Also, original tool's arguments such as the `-u` (user to run the driver as) or `-D` (debug of the driver) do not make sense in the service context — the accounts to use and other arguments to the driver process are part of service setup (and an administrator can manage it there).

Note that while this solution tries to register service instances with same names as NUT configuration sections for the devices, this can not always be possible due to constraints such as syntax supported by a particular service management framework. In this case, the enumerator falls back to MD5 hashes of such section names, and the `upsdrvsvctl` script supports this to map the user-friendly NUT configuration section names to actual service names that it would manage.

References: man pages: [nutupsdrv\(8\)](#), [upsdrvctl\(8\)](#), [upsdrvsvctl\(8\)](#)

6.2.4 Data server configuration (upsd)

Configure `upsd`, which serves data from the drivers to the clients.

First, edit `upsd.conf` to allow access to your client systems. By default, `upsd` will only listen to `localhost` port 3493/tcp. If you want to connect to it from other machines, you must specify each interface you want `upsd` to listen on for connections, optionally with a port number.

```
LISTEN 127.0.0.1 3493
LISTEN :::1 3493
```

As a special case, `LISTEN * <port>` (with an asterisk) will try to listen on "ANY" IP address for both and IPv6 (`:::0`) and IPv4 (`0.0.0.0`), subject to `upsd` command-line arguments, or system configuration or support. Note that if the system supports IPv4-mapped IPv6 addressing per RFC-3493, and does not allow to disable this mode, then there may be one listening socket to handle both address families.

Note

Refer to the NUT user manual [security chapter](#) for information on how to access and secure `upsd` clients connections.

Next, create `upsd.users`. For now, this can be an empty file. You can come back and add more to it later when it's time to configure `upsdmon` or run one of the management tools.

Do not make either file world-readable, since they both hold access control data and passwords. They just need to be readable by the user you created in the preparation process.

The suggested configuration is to `chown` it to `root`, `chgrp` it to the group you created, then make it readable by the group.

Note

If you installed NUT from source and used `make install-as-root`, or if your distribution packaging did, the sample configuration files would have the suggested ownership and permissions assigned, so if you use e.g. `cp -pf upsd.users.sample upsd.users` (as `root`) to start out with some annotated comments and adapt that to your deployment, the copied files should also get the expected safe permissions.

```
::; chown root:nut upsd.conf upsd.users
::; chmod 0640 upsd.conf upsd.users
```

References: man pages: [upsd.conf\(5\)](#), [upsd.users\(5\)](#), [upsd\(8\)](#)

6.2.5 Starting the data server

Start the network data server:

```
::; upsd
```

Make sure it is able to connect to the driver(s) on your system. A successful run looks like this:

```
# upsd
Network UPS Tools upsd 2.4.1
listening on 127.0.0.1 port 3493
listening on :::1 port 3493
Connected to UPS [eaton]: usbhid-ups-eaton
```

`upsd` prints dots while it waits for the driver to respond. Your system may print more or less depending on how many drivers you have and how fast they are.

Note

If `upsd` says that it can't connect to a UPS or that the data is stale, then your `ups.conf` is not configured correctly, or you have a driver that isn't working properly. You must fix this before going on to the next step.

Note

Normally `upsd` requires that at least one driver section is defined in the `ups.conf` file, and refuses to start otherwise. If you intentionally do not have any driver sections defined (yet) but still want the data server to run, respond and report zero devices (e.g. on an automatically managed monitoring deployment), you can enable the `ALLOW_NO_DEVICE true` option in the `upsd.conf` file.

Note

Normally `upsd` requires that at all `LISTEN` directives defined in the `upsd.conf` file are honoured (except for mishaps possible with many names of `localhost`), and refuses to start otherwise. If you want to allow start-up in cases where at least one but possibly not all of the `LISTEN` directives were honoured, you can enable the `ALLOW_NOT_ALL_LISTENERS true` option in the `upsd.conf` file. Note you would have to restart `upsd` to pick up the `LISTEN`'ed IP address if it appears later, so probably configuring `LISTEN *` is a better choice in such cases.

On operating systems with service management frameworks, the data server life-cycle is managed by `nut-server` service.

Reference: man page: [upsd\(8\)](#)

6.2.6 Check the UPS data

Status data

Make sure that the UPS is providing good status data. You can use the `upsc` command-line client for this:

```
:: upsc myupsname@localhost ups.status
```

You should see just one line in response:

```
OL
```

OL means your system is running on line power. If it says something else (like OB — on battery, or LB — low battery), your driver was probably misconfigured during the [Driver configuration](#) step. If you reconfigure the driver, use `upsdrvctl stop` to stop it, then start it again as shown in the [Starting driver\(s\)](#) step.

Reference: man page: [upsc\(8\)](#)

All data

Look at all of the status data which is being monitored.

```
:: upsc myupsname@localhost
```

What happens now depends on the kind of device and driver you have. In the list, you should see `ups.status` with the same value you got above. A sample run on an UPS (Eaton Ellipse MAX 1100) looks like this:

```
battery.charge: 100
battery.charge.low: 20
battery.runtime: 2525
battery.type: PbAc
device.mfr: EATON
device.model: Ellipse MAX 1100
```

```
device.serial: ADKK22008
device.type: ups
driver.name: usbhid-ups
driver.parameter.pollfreq: 30
driver.parameter.pollinterval: 2
driver.parameter.port: auto
driver.version: 2.4.1-1988:1990M
driver.version.data: MGE HID 1.12
driver.version.internal: 0.34
input.sensitivity: normal
input.transfer.boost.low: 185
input.transfer.high: 285
input.transfer.low: 165
input.transfer.trim.high: 265
input.voltage.extended: no
outlet.1.desc: PowerShare Outlet 1
outlet.1.id: 2
outlet.1.status: on
outlet.1.switchable: no
outlet.desc: Main Outlet
outlet.id: 1
outlet.switchable: no
output.frequency.nominal: 50
output.voltage: 230.0
output.voltage.nominal: 230
ups.beeper.status: enabled
ups.delay.shutdown: 20
ups.delay.start: 30
ups.firmware: 5102AH
ups.load: 0
ups.mfr: EATON
ups.model: Ellipse MAX 1100
ups.power.nominal: 1100
ups.productid: ffff
ups.serial: ADKK22008
ups.status: OL CHRG
ups.timer.shutdown: -1
ups.timer.start: -1
ups.vendorid: 0463
```

Reference: man page: [upsc\(8\)](#), [NUT command and variable naming scheme](#)

6.2.7 Startup scripts

Note

This step is not necessary if you installed from packages.

Edit your startup scripts, and make sure `upsdrcvtl` and `upsd` are run every time your system starts. In newer versions of NUT, you may have a `nut.conf` file which sets the `MODE` variable for bundled init-scripts, to facilitate enabling of certain features in the specific end-user deployments.

If you installed from source, check the `scripts` directory for reference init-scripts, as well as `systemd` or `SMF` service methods and manifests.

6.3 Configuring automatic shutdowns for low battery events

The whole point of UPS software is to bring down the OS cleanly when you run out of battery power. Everything else is roughly eye candy.

To make sure your system shuts down properly, you will need to perform some additional configuration and run `upsmon`. Here are the basics.

6.3.1 Shutdown design

When your UPS batteries get low, the operating system needs to be brought down cleanly. Also, the UPS load should be turned off so that all devices that are attached to it are forcibly rebooted, and subsequently start in the predictable order and state suitable for your data center.

Here are the steps that occur when a critical power event happens, for the simpler case of one UPS device feeding one or several systems:

1. The UPS goes on battery
2. The UPS reaches low battery (a "critical" UPS), that is to say, `upsc` displays:

```
ups.status: OB LB
```

The exact behavior depends on the specific device, and is related to such settings and readings as:

- `battery.charge` and `battery.charge.low`
- `battery.runtime` and `battery.runtime.low`

3. The `upsmon` primary notices the "critical UPS" situation and sets "FSD" — the "forced shutdown" flag to tell all secondary systems that it will soon power down the load.

Warning



By design, since we require power-cycling the load and don't want some systems to be powered off while others remain running if the "wall power" returns at the wrong moment as usual, the "FSD" flag can not be removed from the data server unless its daemon is restarted. If we do take the first step in critical mode, then we intend to go all the way — shut down all the servers gracefully, and power down the UPS.

Keep in mind that some UPS devices and corresponding drivers would latch the "FSD" again even if "wall power" is available, but the remaining battery charge is below a threshold configured as "safe" in the device (usually if you manually power on the UPS after a long power outage). This is by design of respective UPS vendors, since in such situation they can not guarantee that if a new power outage happens, their UPS would safely shut down your systems again. So it is deemed better and safer to stay dark until batteries become sufficiently charged.

(If you have no secondary systems, skip to step 6)

4. `upsmon` secondary systems see "FSD" and:
 - generate a `NOTIFY_SHUTDOWN` event
 - wait `FINALDELAY` seconds — typically 5
 - call their `SHUTDOWNCMD`
 - disconnect from `upsd`
 5. The `upsmon` primary system waits up to `HOSTSYNC` seconds (typically 15) for the secondary systems to disconnect from `upsd`. If any are still connected after this time, `upsmon` primary stops waiting and proceeds with the shutdown process.
 6. The `upsmon` primary:
 - generates a `NOTIFY_SHUTDOWN` event
-

- waits `FINALDELAY` seconds — typically 5
 - creates the `POWERDOWNFLAG` file in its local filesystem — usually `/etc/killpower`, or `/run/nut/killpower` in a temporary file system
 - calls the `SHUTDOWNCMD`
7. On most systems, `init` takes over, kills your processes, syncs and unmounts some filesystems, and remounts some read-only.
 8. `init` then runs your shutdown script. This checks for the `POWERDOWNFLAG`, finds it, and tells the UPS driver(s) to power off the load by sending commands to the connected UPS device(s) they manage.
 9. All the systems lose power.
 10. Time passes. The power returns, and the UPS switches back on.
 11. All systems reboot and go back to work.

6.3.2 How you set it up

NUT user creation

Create a `upsd` user for `upsmon` to use while monitoring this UPS.

Edit `upsd.users` and create a new section. The `upsmon` will connect to `upsd` and use these user name (in brackets) and password to authenticate (as specified in its configuration via `MONITOR` line).

This example is for defining a user called "monuser":

```
[monuser]
    password = mypass
    upsmon primary
    # or upsmon secondary
```

References: [upsd\(8\)](#), [upsd.users\(5\)](#)

Reloading the data server

Reload `upsd`. Depending on your configuration, you may be able to do this without stopping the `upsd` daemon process (if it had saved a PID file earlier):

```
;; upsd -c reload
```

If that doesn't work (check the syslog), just restart it:

```
;; upsd -c stop
;; upsd
```

For systems with integrated service management (Linux `systemd`, illumos/Solaris SMF) their corresponding `reload` or `refresh` service actions should handle this as well. Note that such integration generally forgoes saving of PID files, so `upsd -c <cmd>` would not work. If your workflow requires to manage these daemons beside the OS provided framework, you can customize it to start `upsd -FF` and save the PID file.

NUT releases after 2.8.0 define aliases for these units, so if your Linux distribution uses NUT-provided unit definitions, `systemctl reload upsd` may also work.

Note

If you want to make reloading work later, see the entry in the [FAQ](#) about starting `upsd` as a different user.

Power Off flag file

Set the `POWERDOWNFLAG` location for `upsmon`.

In `upsmon.conf`, add a `POWERDOWNFLAG` directive with a filename. The `upsmon` will create this file when the UPS needs to be powered off during a power failure when low battery is reached.

We will test for the presence of this file in a later step.

```
POWERDOWNFLAG /etc/killpower
```

References: man pages: [upsmon\(8\)](#), [upsmon.conf\(5\)](#)

Securing `upsmon.conf`

The recommended setting is to have it owned by `root:nut`, then make it readable by the group and not by the world. This file contains passwords that could be used by an attacker to start a shutdown, so keep it secure.

Note

If you installed NUT from source and used `make install-as-root`, or if your distribution packaging did, the sample configuration files would have the suggested ownership and permissions assigned, so if you use e.g. `cp -pf upsmon.conf.sample upsmon.conf` (as `root`) to start out with some annotated comments and adapt that to your deployment, the copied files should also get the expected safe permissions.

```
:: chown root:nut upsmon.conf
:: chmod 0640 upsmon.conf
```

This step has been placed early in the process so you secure this file before adding sensitive data in the next step.

Create a `MONITOR` directive for `upsmon`

Edit `upsmon.conf` and create a `MONITOR` line with the UPS definition (`<upsname>@<hostname>`), username and password from the [NUT user creation](#) step, and the "primary" or "secondary" setting.

If this system is the UPS manager (i.e. it's connected to this UPS directly and can manage it using a suitable NUT driver), its `upsmon` is the primary:

```
MONITOR myupsname@mybox 1 monuser mypass primary
```

If it's just monitoring this UPS over the network, and some other system is the primary, then this one is a secondary:

```
MONITOR myupsname@mybox 1 monuser mypass secondary
```

The number 1 here is the "power value". This should always be set to 1, unless you have a very special (read: expensive) system with redundant power supplies. In such cases, refer to the User Manual:

- [typical setups for big servers](#),
- [typical setups for data rooms](#).

Note that the "power value" may also be 0 for a monitoring (administrative) system which only observes the remote UPS status but is not impacted by its power events, and so does not shut down when the UPS does.

References: [upsmon\(8\)](#), [upsmon.conf\(5\)](#)

Define a SHUTDOWNCMD for upsmon

Still in *upsmon.conf*, add a directive that tells *upsmon* how to shut down your system. This example seems to work on most systems:

```
SHUTDOWNCMD "/sbin/shutdown -h +0"
```

Notice the presence of "quotes" here to keep it together.

If your system has special needs (e.g. system-provided shutdown handler is ungracefully time constrained), you may want to set this to a script which does customized local shutdown tasks before calling *init* or *shutdown* programs to handle the system side of this operation.

Start upsmon

```
;; upsmon
```

If it complains about something, then check your configuration.

On operating systems with service management frameworks, the monitoring client life-cycle is managed by *nut-monitor* service.

Checking upsmon

Look for messages in the *syslog* to indicate success. It should look something like this:

```
May 29 01:11:27 mybox upsmon[102]: Startup successful
May 29 01:11:28 mybox upsd[100]: Client monuser@192.168.50.1
logged into UPS [myupsname]
```

Any errors seen here are probably due to an error in the config files of either *upsmon* or *upsd*. You should fix them before continuing.

Startup scripts

Note

This step is not need if you installed from packages.

Edit your startup scripts, and add a call to *upsmon*.

Make sure *upsmon* starts when your system comes up. On systems with *upsmon* primary (also running the data server), do it after *upsdrvctl* and *upsd*, or it will complain about not being able to contact the server.

You may delete the *POWERDOWNFLAG* in the startup scripts, but it is not necessary. *upsmon* will clear that file for you when it starts.

Note

Init script examples are provide in the *scripts* directory of the NUT source tree, and in the various [packages](#) that exist.

Shutdown scripts

Note

This step is not need if you installed from packages.

Edit your shutdown scripts, and add `upsdrvctl shutdown`.

You should configure your system to power down the UPS after the filesystems are remounted read-only. Have it look for the presence of the `POWERDOWNFLAG` (from `upsmon.conf(5)`), using this as an example:

```
if (/sbin/upsmon -K)
then
    echo "Killing the power, bye!"
    /sbin/upsdrvctl shutdown

    sleep 120

    # uh oh... the UPS power-off failed
    # you probably want to reboot here so you don't get stuck!
    # *** see also the section on power races in the FAQ! ***
fi
```

A more elaborate example can be found in NUT sources, e.g.: <https://github.com/networkupstools/nut/blob/master/scripts/-systemd/nutshutdown.in>

Warning



- Be careful that `upsdrvctl shutdown` command will probably power off your machine and others fed by the UPS(es) which it manages. Don't use it unless your system is ready to be halted by force. If you run RAID, read the [RAID warning](#) below!
 - Make sure the filesystem(s) containing `upsdrvctl`, `upsmon`, the `POWERDOWNFLAG` file, `ups.conf` and your UPS driver(s) are mounted (possibly in read-only mode) when the system gets to this point. Otherwise it won't be able to figure out what to do.
 - If for some reason you can not ensure `upsmon` program is executable at this point, your script can `(test -f /etc/killpower)` in a somewhat non-portable manner, instead of asking `upsmon -K` for the verdict according to its current configuration.
-

Testing shutdowns

UPS equipment varies from manufacturer to manufacturer and even within model lines. You should test the [shutdown sequence](#) on your systems before leaving them unattended. A successful sequence is one where the OS halts before the battery runs out, and the system restarts when power returns.

The first step is to see how `upsdrvctl` will behave without actually turning off the power. To do so, use the `-t` argument:

```
:: upsdrvctl -t shutdown
```

It will display the sequence without actually calling the drivers.

You can finally test a forced shutdown sequence (FSD) using:

```
:: upsmon -c fsd
```

This will execute a full shutdown sequence, as presented in [Shutdown design](#), starting from the 3rd step.

If everything works correctly, the computer will be forcibly powered off, may remain off for a few seconds to a few minutes (depending on the driver and UPS type), then will power on again.

If your UPS just sits there and never resets the load, you are vulnerable to a power race and should add the "reboot after timeout" hack at the very least.

Also refer to the section on power races in the [FAQ](#).

6.3.3 Using suspend to disk

Support for suspend to RAM and suspend to disk has been available in the Linux kernel for a while now. For obvious reasons, suspending to RAM isn't particularly useful when the UPS battery is getting low, but suspend to disk may be an interesting concept.

This approach minimizes the amount of disruption which would be caused by an extended outage. The UPS goes on battery, then reaches low battery, and the system takes a snapshot of itself and halts. Then it is turned off and waits for the power to return.

Once the power is back, the system reboots, pulls the snapshot back in, and keeps going from there. If the user happened to be away when it happened, they may return and have no idea that their system actually shut down completely in the middle (although network connections will drop).

In order for this to work, you need to shutdown NUT (UPS driver, `upsd` server and `upsmon` client) in the `suspend` script and start them again in the `resume` script. Don't try to keep them running. The `upsd` server will latch the FSD state (so it won't be usable after resuming) and so will the `upsmon` client. Some drivers may work after resuming, but many don't and some UPS devices will require re-initialization, so it's best not to keep them running either.

Note

Starting with NUT v2.8.3, there is some growing support for system-wide sleep on some platforms (e.g. to catch the "going to sleep" event and make a note of it in the daemons, to take the least-surprise corrective actions after a significant change in system clock readings), but the warnings in previous paragraph may still apply.

After stopping NUT driver, server and client you'll have to send the UPS the command to shutdown only if the `POWERDOWNFLAG` is present. Note that most likely you'll have to allow for a grace period after calling `upsdrcvtl shutdown` since the system will still have to take a snapshot of itself after that. Not all drivers and devices support this, so before going down this road, make sure that the one you're using does.

- see if you can query or configure settings named like `load.off.delay`, `ups.delay.shutdown`, `offdelay` and/or `shutdown_delay`

6.3.4 RAID warning

If you run any sort of RAID equipment, make sure your arrays are either halted (if possible) or switched to "read-only" mode. Otherwise you may suffer a long resync once the system comes back up.

The kernel may not ever run its final shutdown procedure, so you must take care of all array shutdowns in userspace before `upsdrcvtl shutdown` runs.

If you use software RAID (md) on Linux, get `mdadm` and try using `mdadm --readonly` to put your arrays in a safe state. This has to happen after your shutdown scripts have remounted the filesystems.

On hardware RAID or other kernels, you have to do some detective work. It may be necessary to contact the vendor or the author of your driver to find out how to put the array in a state where a power loss won't leave it "dirty".

Our understanding is that most if not all RAID devices on Linux will be fine unless there are pending writes. Make sure your filesystems are remounted read-only and you should be covered.

6.4 Typical setups for enterprise networks and data rooms

The split nature of this UPS monitoring software allows a wide variety of power connections. This chapter will help you identify how things should be configured using some general descriptions.

There are two main elements:

1. There's a UPS attached to a communication (serial, USB or network) port on this system.
2. This system depends on a UPS for power.

You can play "mix and match" with those two to arrive at these descriptions for individual hosts:

- A: 1 but not 2
- B: 2 but not 1
- C: 1 and 2

A small to medium sized data room usually has one *C* and a bunch of *B*s. This means that there's a system (type *C*) hooked to the UPS which depends on it for power. There are also some other systems in there (type *B*) which depend on that same UPS for power, but aren't directly connected to it communications-wise.

Larger data rooms or those with multiple UPSes may have several "clusters" of the "single *C*, many *B*s" depending on how it's all wired.

Finally, there's a special case. Type *A* systems are connected to an UPS's communication port, but don't depend on it for power. This usually happens when an UPS is physically close to a box and can reach the serial port, but the power wiring is such that it doesn't actually feed that box.

Once you identify a system's type, use this list to decide which of the programs need to be run for monitoring:

- A: driver and `upsd`
- B: `upsmon` (in secondary mode)
- C: driver, `upsd`, and `upsmon` (in primary mode, as the UPS manager)



To further complicate things, you can have a system that is hooked to multiple UPSes, but only depends on one for power. This particular situation makes it an *A* relative to one UPS, and a *C* relative to the other. The software can handle this — you just have to tell it what to do.

Note

NUT can also serve as a data proxy to increase the number of clients, or share the communication load between several `upsd` instances.

If you are running large server-class systems that have more than one power feed, see the next section for information on how to handle it properly.

6.5 Typical setups for big servers with UPS redundancy

By using multiple `MONITOR` statements in `upsmon.conf`, you can configure an environment where a large machine with redundant power monitors multiple separate UPSes.



6.5.1 Example configuration

For the examples in this section, we will use a server with four power supplies installed and locally running the full NUT stack, including `upsmon` in primary mode — as the UPS manager.

Two UPSes, *Alpha* and *Beta*, are each driving two of the power supplies (by adding up, we know about the four power supplies of the current system). This means that either *Alpha* or *Beta* can totally shut down and the server will be able to keep running.

The `upsmon.conf` configuration which reflects this is the following:

```
MONITOR ups-alpha@myhost 2 monuser mypass primary
MONITOR ups-beta@myhost 2 monuser mypass primary
MINSUPPLIES 2
```

With such configuration, `upsmon` on this system will only shut down when both UPS devices reach a critical (on battery + low battery) condition, since *Alpha* and *Beta* each provide the same power value.

As an added bonus, this means you can move a running server from one UPS to another (for maintenance purpose for example) without bringing it down since the minimum sufficient power will be provided at all times.

The `MINSUPPLIES` line tells `upsmon` that we need at least 2 power supplies to be receiving power from a good UPS (on line or on battery, just not on battery **and** low battery).

Note

We could have used a *Power Value* of 1 for both UPS, and have `MINSUPPLIES` set to 1 too. These values are purely arbitrary, so you are free to use your own rules. Here, we have linked these values to the number of power supplies that each UPS is feeding (2) since this maps better to physical topology and allows to throw a third or fourth UPS into the mix without much configuration headache.

6.5.2 Multiple UPS shutdowns ordering

If you have multiple UPSes connected to your system, chances are that you need to shut them down in a specific order. The goal is to shut down everything but the one keeping `upsmon` alive at first, then you do that one last.

To set the order in which your UPSes receive the shutdown commands, define the `sdorder` value in your `ups.conf` device sections.

```
[bigone]
    driver = usbhid-ups
    port = auto
    sdorder = 2
```

```
[littleguy]
    driver = mge-shut
    port = /dev/ttyS0
    sdorder = 1

[misc]
    driver = blazer_ser
    port = /dev/ttyS1
    sdorder = 0
```

The order runs from 0 to the highest number available. So, for this configuration, the order of shutdowns would be *misc*, *littleguy*, and then *bigone*.

Note

If you have a UPS that shouldn't be powered off when running `upsdrvctl shutdown`, set its `sdorder` to `-1`.

6.5.3 Other redundancy configurations

There are a lot of ways to handle redundancy and they all come down to how many power supplies, power cords and independent UPS connections you have. A system with a 1:1 cord:supply ratio has more wires stuffed behind it, but it's much easier to move things around since any given UPS drives a smaller percentage of the overall power.

More information can be found in the [NUT user manual](#), and the various [user manual pages](#).

7 Advanced usage and scheduling notes

upsmon can call out to a helper script or program when the device changes state. The example `upsmon.conf` has a full list of which state changes are available — ONLINE, ONBATT, LOWBATT, and more.

There are two options, that will be presented in details:

- the simple approach: create your own helper, and manage all events and actions yourself,
- the advanced approach: use the NUT provided helper, called *upssched*.

7.1 The simple approach, using your own script

7.1.1 How it works relative to upsmon

Your command will be called with the full text of the message as one argument.

For the default values, refer to the sample `upsmon.conf` file.

The environment string `NOTIFYTYPE` will contain the type string of whatever caused this event to happen — ONLINE, ONBATT, LOWBATT, ...

Making this some sort of shell script might be a good idea, but the helper can be in any programming or scripting language.

Note

Remember that your helper must be **executable**. If you are using a script, make sure the execution flags are set.

For more information, refer to [upsmon\(8\)](#) and [upsmon.conf\(5\)](#) manual pages.

7.1.2 Setting up everything

- Set EXEC flags on various things in `upsmon.conf(5)`:

```
NOTIFYFLAG ONBATT EXEC
NOTIFYFLAG ONLINE EXEC
```

If you want other things like WALL or SYSLOG to happen, just add them:

```
NOTIFYFLAG ONBATT EXEC+WALL+SYSLOG
```

You get the idea.

- Tell upsmon where your script is

```
NOTIFYCMD /path/to/my/script
```

- Make a simple script like this at that location:

```
#!/bin/bash
echo "$*" | sendmail -F"ups@mybox" bofh@pager.example.com
```

- Restart upsmon, pull the plug, and see what happens.

That approach is bare-bones, but you should get the text content of the alert in the body of the message, since upsmon passes the alert text (from NOTIFYMSG) as an argument.

7.1.3 Using more advanced features

Your helper script will be run with a few environment variables set.

- UPSNAME: the name of the system that generated the change.
This will be one of your identifiers from the MONITOR lines in upsmon.conf.
- NOTIFYTYPE: this will be ONLINE, ONBATT, or whatever event took place which made upsmon call your script.

You can use these to do different things based on which system has changed state. You could have it only send pages for an important system while totally ignoring a known trouble spot, for example.

7.1.4 Suppressing notify storms

upsmon will call your script every time an event happens that has the EXEC flag set. This means a quick power failure that lasts mere seconds might generate a notification storm. To suppress this sort of annoyance, use upssched as your NOTIFYCMD program, and configure it to call your command after a timer has elapsed.

7.2 The advanced approach, using upssched

upssched is a helper for upsmon that will invoke commands for you at some interval relative to a UPS event. It can be used to send pages, mail out notices about things, or even shut down the box early.

There will be examples scattered throughout. Change them to suit your pathnames, UPS locations, and so forth.

7.2.1 How upssched works relative to upsmon

When an event occurs, upsmon will call whatever you specify as a *NOTIFYCMD* in your upsmon.conf, if you also enable the *EXEC* in your *NOTIFYFLAGS*. In this case, we want upsmon to call upssched as the notifier, since it will be doing all the work for us. So, in the upsmon.conf:

```
NOTIFYCMD /usr/local/ups/sbin/upssched
```

Then we want upsmon to actually *use* it for the notify events, so again in the upsmon.conf we set the flags:

```
NOTIFYFLAG ONLINE SYSLOG+EXEC
NOTIFYFLAG ONBATT SYSLOG+WALL+EXEC
NOTIFYFLAG LOWBATT SYSLOG+WALL+EXEC
... and so on.
```

For the purposes of this document I will only use those three, but you can set the flags for any of the valid notify types.

7.2.2 Setting up your upssched.conf

Once upsmon has been configured with the NOTIFYCMD and EXEC flags, you're ready to deal with the upssched.conf details. In this file, you specify just what will happen when a given event occurs on a particular UPS.

First you need to define the name of the script or program that will handle timers that trigger. This is your CMDSCRIPT, and needs to be above any AT defines. There's an example provided with the program, so we'll use that here:

```
CMDSCRIPT /usr/local/ups/bin/upssched-cmd
```

Then you have to define the variables PIPEFN and LOCKFN; the former sets the file name of the FIFO that will pass communications between processes to start and stop timers, while the latter sets the file name for a temporary file created by upssched in order to avoid a race condition under some circumstances. Please see the relevant comments in upssched.conf for additional information and advice about these variables.

Now you can tell your CMDSCRIPT what to do when it is called by upsmon.

The big picture

The design in a nutshell is:

```
upsmon ---> calls upssched ---> calls your CMDSCRIPT
```

Ultimately, the CMDSCRIPT does the actual useful work, whether that's initiating an early shutdown with *upsmon -c fsd*, sending a page by calling sendmail, or opening a subspace channel to V'ger.

Establishing timers

Let's say that you want to receive a notification when any UPS has been running on battery for 30 seconds. Create a handler that starts a 30 second timer for an ONBATT condition.

```
AT ONBATT * START-TIMER onbattwarn 30
```

This means "when any UPS (the *) goes on battery, start a timer called onbattwarn that will trigger in 30 seconds". We'll come back to the onbattwarn part in a moment. Right now we need to make sure that we don't trigger that timer if the UPS happens to come back before the time is up. In essence, if it goes back on line, we need to cancel it. So, let's tell upssched that.

```
AT ONLINE * CANCEL-TIMER onbattwarn
```

Note

Timers are pure in-memory mechanisms, specific to upssched. Conversely to other mechanisms found in NUT, such as upsmon→POWERDOWNFLAG, there is no file created on the filesystem.

Executing commands immediately

As an example, consider the scenario where a UPS goes onto battery power. However, the users are not informed until 30 seconds later—using a timer as described above. Whilst this may let the **logged in** users know that the UPS is on battery power, it does not inform any users subsequently logging in. To enable this we could, at the same time, create a file which is read and displayed to any user trying to login whilst the UPS is on battery power. If the UPS comes back onto utility power within 30 seconds, then we can cancel the timer and remove the file, as described above. However, if the UPS comes back onto utility power say 5 minutes later then we do not want to use any timers but we still want to remove the file. To do this we could use:

```
AT ONLINE * EXECUTE ups-back-on-power
```

This means that when upsmmon detects that the UPS is back on utility power it will signal upssched. Upssched will see the above command and simply pass *ups-back-on-power* as an argument directly to CMDSCRIPT. This occurs immediately, there are no timers involved.

7.2.3 Writing the command script handler

OK, now that upssched knows how the timers are supposed to work, let's give it something to do when one actually triggers. The name of the example timer is *onbattwarn*, so that's the argument that will be passed into your CMDSCRIPT when it triggers. This means we need to do some shell script writing to deal with that input.

```
#!/bin/sh

case $1 in
    onbattwarn)
        # Send a notification mail
        echo "The UPS has been on battery for awhile" \
        | mail -s"UPS monitor" bofh@pager.example.com
        # Create a flag-file on the filesystem, for your own processing
        /usr/bin/touch /some/path/ups-on-battery
        ;;
    ups-back-on-power)
        # Delete the flag-file on the filesystem
        /bin/rm -f /some/path/ups-on-battery
        ;;
    *)
        logger -t upssched-cmd "Unrecognized command: $1"
        ;;
esac
```

This is a very simple script example, but it shows how you can test for the presence of a given trigger. With multiple ATs creating various timer names, you will need to test for each possibility and handle it according to your desires.

Note

You can invoke just about anything from inside the CMDSCRIPT. It doesn't need to be a shell script, either—that's just an example. If you want to write a program that will parse `argv[1]` and deal with the possibilities, that will work too.

7.2.4 Early Shutdowns

One thing that gets requested a lot is early shutdowns in upsmmon. With upssched, you can now have this functionality. Just set a timer for some length of time at ONBATT which will invoke a shutdown command if it elapses. Just be sure to cancel this timer if you go back ONLINE before then.

The best way to do this is to use the upsmmon callback feature. You can make upsmmon set the "forced shutdown" (FSD) flag on the upsd so your secondary systems shut down early too. Just do something like this in your CMDSCRIPT:

```
/sbin/upsmon -c fsd
```

Note

the path to `upsmon` must be provided. The default for an installation built from sources is `/usr/local/ups` (so `/usr/local/ups/sbin/upsmon`), while packaged installations will generally comply to [FHS—Filesystem Hierarchy Standard](#) (so `/sbin/upsmon`).

It's not a good idea to call your system's shutdown routine directly from the `CMDSCRIPT`, since there's no synchronization with the secondary systems hooked to the same UPS. FSD is the primary's way of saying "we're shutting down **now** like it or not, so you'd better get ready".

7.2.5 Background

This program was written primarily to fulfill the requests of users for the early shutdown scenario. The "outboard" design of the program (relative to `upsmon`) was intended to reduce the load on the average system. Most people don't have the requirement of shutting down after `n` seconds on battery, since the usual `OB+LB` testing is sufficient.

This program was created separately so those people don't have to spend CPU time and RAM on something that will never be used in their environments.

The design of the timer handler is also geared towards minimizing impact. It will come and go from the process list as necessary. When a new timer is started, a process will be forked to actually watch the clock and eventually start the `CMDSCRIPT`. When a timer triggers, it is removed from the queue. Canceling a timer will also remove it from the queue. When no timers are present in the queue, the background process exits.

This means that you will only see `upssched` running when one of two things is happening:

1. There's a timer of some sort currently running
2. `upsmon` just called it, and you managed to catch the brief instance

The final optimization handles the possibility of trying to cancel a timer when there's none running. If there's no process already running, there are no timers to cancel, and furthermore there is no need to start a clock-watcher. As a result, it skips that step and exits sooner.

8 NUT outlets management and PDU notes

NUT supports advanced outlets management for any kind of device that proposes it. This chapter introduces how to manage outlets in general, and how to take advantage of the provided features.

8.1 Introduction

Outlets are the core of Power Distribution Units. They allow you to turn on, turn off or cycle the load on each outlet.

Some UPS models also provide manageable outlets (Eaton, MGE, Powerware, Tripplite, ...) that help save power in various ways, and manage loads more intelligently.

Finally, some devices can be managed in a PDU-like way. Consider blade systems: the blade chassis can be controlled remotely to turn on, turn off or cycle the power on individual blade servers.

NUT allows you to control all these devices!

8.2 NUT outlet data collection

NUT provides a complete and uniform integration of outlets related data, through the *outlet* collection.

First, there is a special outlet, called *main outlet*. You can access it through *outlet.{id, desc, ...}* without any index.

Any modification through the *main outlet* will affect **all** outlets. For example, calling the command *outlet.load.cycle* will cycle all outlets.

Next, outlets index starts from **1**. Index *0* is implicitly reserved to the *main outlet*. So the first outlet is *outlet.1.**.

For a complete list of outlet data and commands, refer to the [NUT command and variable naming scheme](#).

An example upsc output (data/epdu-managed.dev) is available in the source archive.

Note

The variables supported depend on the exact device type.

8.3 Outlets on PDU

Smart Power Distribution Units provide at least various meters, related to current, power and voltage.

Some more advanced devices also provide control through the *load.off*, *load.on* and *load.cycle* commands.

8.4 Outlets on UPS

Some advanced Uninterruptible Power Supplies provide smart outlet management.

This allows to program a limited backup time to non-critical loads in order to keep the maximum of the battery reserve for critical equipment.

This also allows the same remote electrical management of devices provided by PDUs, which can be very interesting in Data Centers.

For example, on small setup, you can plug printers, USB devices, hubs, (...) into managed outlets. Depending on your UPS's capabilities, you will be able to turn off those loads:

- after some minutes of back-up time using *outlet.n.delay.start*,
- when reaching a percentage battery charge using *outlet.n.autoswitch.charge.low*.

This will ensure a maximum runtime for the computer.

On bigger systems, with bigger UPSs, this is the same thing with servers instead of small devices.

Note

If you need the scheduling function and your device doesn't support it, you can still use [NUT scheduling features](#).



Warning

don't plug the UPS's communication cable (USB or network) on a managed outlet. Otherwise, all computers will be stopped as soon as the communication is lost.

8.5 Other type of devices

As mentioned in the introduction, some other devices can be considered and managed like PDUs. This is the case in most blade systems, where the blade chassis offers power management services.

This way, you can control remotely each blade server as if it were a PDU outlet.

This category of devices is generally called Remote Power Controls — or "RPC" in NUT.

9 NUT daisychain support notes

NUT supports daisychained devices for any kind of device that proposes it. This chapter introduces:

- for developers: how to implement such mechanism,
- for users: how to manage and use daisychained devices in NUT in general, and how to take advantage of the provided features.

9.1 Introduction

It's not unusual to see some daisy-chained PDUs or UPSs, connected together in master-slave mode, to only consume 1 IP address for their communication interface (generally, network card exposing SNMP data) and expose only one point of communication to manage several devices, through the daisy-chain master.

This breaks the historical consideration of NUT that one driver provides data for one unique device. However, there is an actual need, and a smart approach was considered to fulfill this, while not breaking the standard scope (for base compatibility).

9.2 Implementation notes

9.2.1 General specification

The daisychain support uses the device collection to extend the historical NUT scope (1 driver — 1 device), and provides data from the additional devices accessible through a single management interface.

A new variable was introduced to provide the number of devices exposed: the `device.count`, which:

- defaults to 1
- if higher than 1, enables daisychain support and access to data of each individual device through `device.X.{...}`

To ensure backward compatibility in NUT, the data of the various devices are exposed the following way:

- `device.0` is a special case, for the whole set of devices (the whole daisychain). It is equivalent to `device` (without `.X` index) and root collections. The idea is to be able to get visibility and control over the whole daisychain from a single point.
- daisy-chained devices are available from `device.1` (master) to `device.N` (slaves).

That way, client applications that are unaware of the daisychain support, will only see the whole daisychain, as it would normally seem, and not nothing at all.

Moreover, this solution is generic, and not specific to the ePDU use case currently considered. It thus support both the current NUT scope, along with other use cases (parallel / serial UPS setups), and potential evolution and technology change (hybrid chain with UPS and PDU for example).

Devices status handling

FIXME: To be clarified...

Devices alarms handling

Devices (master and slaves) alarms are published in `device.X.ups.alarm`, which may evolve into `device.X.alarm`. If any of the devices has an alarm, the main `ups.status` will publish an `ALARM` flag. This flag is be cleared once all devices have no alarms anymore.

Note

`ups.alarm` behavior is not yet defined (all devices alarms vs. list of device(s) that have alarms vs. nothing?)

Example

Here is an example excerpt of three PDUs, connected in daisychain mode, with one master and two slaves:

```
device.count: 3
device.mfr: EATON
device.model: EATON daisychain PDU
device.1.mfr: EATON
device.1.model: EPDU MI 38U-A IN: L6-30P 24A 1P OUT: 36XC13:6XC19
device.2.mfr: EATON
device.2.model: EPDU MI 38U-A IN: L6-30P 24A 1P OUT: 36XC13:6XC19
device.3.mfr: EATON
device.3.model: EPDU MI 38U-A IN: L6-30P 24A 1P OUT: 36XC13:6XC19
...
device.3.ups.alarm: high current critical!
device.3.ups.status: ALARM
...
input.voltage: ??? (proposal: range or list or average?)
device.1.input.voltage: 237.75
device.2.input.voltage: 237.75
device.3.input.voltage: 237.75
...
outlet.1.status: ?? (proposal: "on, off, off)
device.1.outlet.1.status: on
device.2.outlet.1.status: off
device.3.outlet.1.status: off
...
ups.status: ALARM
```

9.2.2 Information for developers

Note

These details are dedicated to the `snmp-ups` driver!

In order to enable daisychain support for a range of devices, developers have to do two things:

- Add a `device.count` entry in a mapping file (see `*-mib.c`)
- Modify mapping entries to include a format string for the daisychain index

Optionally, if there is support for outlets and / or outlet-groups, there is already a template formatting string. So you have to tag such templates with multiple definitions, to point if the daisychain index is the first or second formatting string.

Base support

In order to enable daisychain support on a mapping structure, the following steps have to be done:

- Add a "device.count" entry in the mapping file: snmp-ups will determine if the daisychain support has to be enabled (if more than 1 device). To achieve this, use one of the following type of declarations:

a) point at an OID which provides the number of devices:

```
{ "device.count", 0, 1, ".1.3.6.1.4.1.13742.6.3.1.0", "1",
  SU_FLAG_STATIC, NULL },
```

b) point at a template OID to guesstimate the number of devices, by walking through this template, until it fails:

```
{ "device.count", 0, 1, ".1.3.6.1.4.1.534.6.6.7.1.2.1.2.%i", "1",
  SU_FLAG_STATIC, NULL, NULL },
```

- Modify all entries so that OIDs include the formatting string for the daisychain index. For example, if you have the following entry:

```
{ "device.model", ST_FLAG_STRING, SU_INFOSIZE,
  ".1.3.6.1.4.1.534.6.6.7.1.2.1.2.0", ... },
```

And if the last "0" of the the 4th field represents the index of the device in the daisychain, then you would have to adapt it the following way:

```
{ "device.model", ST_FLAG_STRING, SU_INFOSIZE,
  ".1.3.6.1.4.1.534.6.6.7.1.2.1.2.%i", ... },
```

Templates with multiple definitions

If there already exist templates in the mapping structure, such as for single outlets and outlet-groups, you also need to specify the position of the daisychain device index in the OID strings for all entries in the mapping table, to indicate where the daisychain insertion point is exactly.

For example, using the following entry:

```
{ "outlet.%i.current", 0, 0.001, ".1.3.6.1.4.1.534.6.6.7.6.4.1.3.0.%i",
  NULL, SU_OUTLET, NULL, NULL },
```

You would have to translate it to:

```
{ "outlet.%i.current", 0, 0.001, ".1.3.6.1.4.1.534.6.6.7.6.4.1.3.%i.%i",
  NULL, SU_OUTLET | SU_TYPE_DAISSY_1, NULL, NULL },
```

SU_TYPE_DAISSY_1 flag indicates that the daisychain index is the first %i specifier in the OID template string. If it is the second one, use SU_TYPE_DAISSY_2.

Devices alarms handling

Two functions are available to handle alarms on daisychain devices in your driver:

- `device_alarm_init()`: clear the current alarm buffer
- `device_alarm_commit(const int device_number)`: commit the current alarm buffer to device.<device_number> and increase the count of alarms. If the current alarms buffer is empty, the count of alarm is decreased, and the variable device.<device_number>.ups.alarm is removed from publication. Once the alarm count reaches "0", the main (device.0) ups.status will also remove the "ALARM" flag.

Note

When implementing a new driver, the following functions have to be called:

- `alarm_init()` at the beginning of the main update loop, for the whole daisychain. This will set the alarm count to "0", and reinitialize all alarms,
- `device_alarm_init()` at the beginning of the per-device update loop. This will only clear the alarms for the current device,
- `device_alarm_commit()` at the end of the per-device update loop. This will flush the current alarms for the current device,
- also `device_alarm_init()` at the end of the per-device update loop. This will clear the current alarms, and ensure that this buffer will not be considered by other subsequent devices,
- `alarm_commit()` at the end of the main update loop, for the whole daisychain. This will take care of publishing or not the "ALARM" flag in the main `ups.status(device.0, root collection)`.

10 Notes on securing NUT

The NUT Team is very interested in providing the highest security level to its users.

Many internal and external mechanisms exist to secure NUT. And several steps are needed to ensure that your NUT setup meets your security requirements.

This chapter will present you these mechanisms, by increasing order of security level. This means that the more security you need, the more mechanisms you will have to apply.

Note

You may want to have a look at NUT Quality Assurance, since some topics are related to NUT security and reliability.

10.1 How to verify the NUT source code signature

In order to verify the NUT source code signature for releases, perform the following steps:

- Retrieve the **NUT source code** (`nut-X.Y.Z.tar.gz`) and the matching signature (`nut-X.Y.Z.tar.gz.sig`)
- Retrieve the **NUT maintainer's signature keyring**:

```
$ gpg --fetch-keys https://www.networkupstools.org/source/nut-key.gpg
```

Note

As of NUT 2.8.0, a new release key is used, but the `nut-key.gpg` should be cumulative with older chain key files (includes them). You can view the key list in a downloaded copy of the URL above with:

```
$ gpg --with-colons --import-options import-show --dry-run --import < nut-key. ↵
gpg
```

... and as of this writing, it should contain two key sets for various identities of "Arnaud Quette" and one set of "Jim Klimov".

Just in case, the previous key file used since NUT 2.7.3 release is stored as **NUT old maintainer's signature for 2.7.3-2.7.4 releases**

In order to verify an even older release, please use **NUT old maintainer's signature since 2002 until 2.7.3 release**

- Launch the GPG checking using the following command:

```
$ gpg --verify nut-X.Y.Z.tar.gz.sig
```

- You should see a message mentioning a "Good signature", with formatting which depends on your gpg version, like:

```
gpg: Signature made Thu Jun  1 00:10:16 2023 CEST
...
gpg: Good signature from "Jim Klimov ..."
...
Primary key fingerprint: B834 59F7 76B9 0224 988F  36C0 DE01 84DA 7043 DCF7
...
```

Note

The previously used maintainer's signatures would output (with markup of older gpg tools here):

```
gpg: Signature made Wed Apr 15 15:55:30 2015 CEST using RSA key ID 55CA5976
gpg: Good signature from "Arnaud Quette ..."
...
```

or:

```
gpg: Signature made Thu Jul  5 16:15:05 2007 CEST using DSA key ID 204DDF1B
gpg: Good signature from "Arnaud Quette ..."
...
```

10.2 How to verify the NUT source code checksum

As a weaker but simpler alternative to verifying a **signature**, you can verify just the accompanying checksums of the source archive file. This is useful primarily to check against bit-rot in original storage or in transit. As far as disclaimers go: ideally, you should cover all provided algorithms—e.g. MD5 and SHA256—to minimize the chance that intentional malicious tampering on the wire goes undetected. A myriad tools can check that on various platforms; some examples follow:

```
# Example original checksum to compare with, from NUT website:
$ cat nut-2.8.0.tar.gz.sha256
c3e5a708da797b7c70b653d37b1206a000fcb503b85519fe4cdf6353f792bfe5  nut-2.8.0.tar.gz
```

```
# Generate checksum of downloaded archive with perl (a NUT build dependency
# generally, though you may have to install Digest::SHA module from CPAN):
$ perl -MDigest::SHA=sha256_hex -le "print sha256_hex <>" nut-2.8.0.tar.gz
c3e5a708da797b7c70b653d37b1206a000fcb503b85519fe4cdf6353f792bfe5
```

```
# Generate checksum of downloaded archive with openssl (another optional
# NUT build dependency):
$ openssl sha256 nut-2.8.0.tar.gz
SHA256(nut-2.8.0.tar.gz)=  ↵
    c3e5a708da797b7c70b653d37b1206a000fcb503b85519fe4cdf6353f792bfe5
```

```
# Generate checksum of downloaded archive with coreutils:
$ sha256sum nut-2.8.0.tar.gz
c3e5a708da797b7c70b653d37b1206a000fcb503b85519fe4cdf6353f792bfe5  nut-2.8.0.tar.gz
```

```
# Auto-check downloaded checksum against downloaded archive with coreutils:
$ sha256sum -c nut-2.8.0.tar.gz.sha256
nut-2.8.0.tar.gz: OK
```

```
# Generate checksum of downloaded archive with GPG:
$ gpg --print-md SHA256 nut-2.8.0.tar.gz
nut-2.8.0.tar.gz: C3E5A708 DA797B7C 70B653D3 7B1206A0
                  00FCB503 B85519FE 4CDF6353 F792BFE5
```

10.3 System level privileges and ownership

All configuration files should be protected so that the world can't read them. Use the following commands to accomplish this:

```
chown root:nut /etc/nut/*
chmod 640 /etc/nut/*
```

Finally, the [state path](#) directory, which holds the communication between the driver(s) and `upsd`, should also be secured.

```
chown root:nut /var/state/ups
chmod 0770 /var/state/ups
```

10.4 NUT level user privileges

Administrative commands such as setting variables and the instant commands are powerful, and access to them needs to be restricted.

NUT provides an internal mechanism to do so, through [upsd.users\(5\)](#).

This file defines who may access instant commands and settings, and what is available.

During the initial [NUT user creation](#), we have created a monitoring user for `upsmon`.

You can also create an `administrator` user in NUT with full power using:

```
[administrator]
    password = mypass
    actions = set
    instcmds = all
```

For more information on how to restrict actions and instant commands, refer to [upsd.users\(5\)](#) manual page.

Note

NUT administrative user definitions should be used in conjunction with [TCP Wrappers](#).

10.5 Network access control

If you are not using NUT on a standalone setup, you will need to enforce network access to `upsd`.

There are various ways to do so.

10.5.1 NUT LISTEN directive

[upsd.conf\(5\)](#).

```
LISTEN interface port
```

Bind a listening port to the interface specified by its Internet address. This may be useful on hosts with multiple interfaces. You should not rely exclusively on this for security, as it can be subverted on many systems.

Listen on TCP port `port` instead of the default value which was compiled into the code. This overrides any value you may have set with `configure --with-port`. If you don't change it with `configure` or this value, `upsd` will listen on port 3493 for this interface.

Multiple LISTEN addresses may be specified. The default is to bind to `127.0.0.1` if no LISTEN addresses are specified (and `::1` if IPv6 support is compiled in).

```
LISTEN 127.0.0.1
LISTEN 192.168.50.1
LISTEN ::1
LISTEN 2001:0db8:1234:08d3:1319:8a2e:0370:7344
```

As a special case, `LISTEN * <port>` (with an asterisk) will try to listen on "ANY" IP address for both IPv6 (`:::0`) and IPv4 (`0.0.0.0`), subject to `upsd` command-line arguments, or system configuration or support. Note that if the system supports IPv4-mapped IPv6 addressing per RFC-3493, and does not allow to disable this mode, then there may be one listening socket to handle both address families.

This parameter will only be read at startup. You'll need to restart (rather than reload) `upsd` to apply any changes made here.

10.5.2 Firewall

NUT has its own official IANA port: 3493/tcp.

The `upsmon` process on secondary systems, as well as any other NUT client (such as `upsc`, `upscmd`, `upsw`, NUT-Monitor, ...) connects to the `upsd` process on the system which manages the UPS, via this TCP port. Usually an `upsmon` process runs on the latter system in "primary" mode for the devices connected to it.

The `upsd` process does not initiate outgoing connections.

Certain NUT drivers (for network-managed devices) can initiate their own connections to various ports according to corresponding vendor protocol.

You should use this to restrict network access.

Uncomplicated Firewall (UFW) support

NUT can tightly integrate with [Uncomplicated Firewall](#) using the provided profile (`nut.ufw.profile`).

You must first install the profile on your system:

```
$ cp nut.ufw.profile /etc/ufw/applications.d/
```

To enable outside access to your local `upsd`, use:

```
$ ufw allow NUT
```

To restrict access to the network `192.168.X.Y`, use:

```
$ ufw allow from 192.168.0.0/16 to any app NUT
```

You can also use graphical frontends, such as `gui-ufw` (`gufw`), `ufw-kde` or `ufw-frontends`.

For more information, refer to:

- [UFW homepage](#),
- [UFW project page](#),
- [UFW wiki](#),
- UFW manual page, section APPLICATION INTEGRATION

10.5.3 TCP Wrappers

If the server is build with tcp-wrappers support enabled, it will check if the NUT username is allowed to connect from the client address through the `/etc/hosts.allow` and `/etc/hosts.deny` files.

Note

This will only be done for commands that require the user to be logged into the server.

```
hosts.allow:
upsd : admin@127.0.0.1/32
upsd : observer@127.0.0.1/32 observer@192.168.1.0/24

hosts.deny:
upsd : ALL
```

Further details are described in `hosts_access(5)`.

10.6 Configuring SSL

SSL is available as a build option (`--with-ssl`).

It encrypts sessions between `upsd` and clients, and can also be used to authenticate servers.

This means that stealing port 3493 from `upsd` will no longer net you interesting passwords.

Several things must happen before this will work, however. This chapter will present these steps.

SSL is available via two back-end libraries : NSS and OpenSSL (historically). You can choose to use one of them by specifying it with a build option (`--with-nss` or `--with-openssl`). If neither is specified, the configure script will try to detect one of them, with a precedence for OpenSSL.

10.6.1 OpenSSL backend usage

This section describes how to enable NUT SSL support using **OpenSSL**.

Install OpenSSL

Install **OpenSSL** as usual, either from source or binary packages. If using binary packages, be sure to include the developer libraries.

Recompile and install NUT

Recompile NUT from source, starting with `configure --with-openssl`.

Then install everything as usual.

Create a certificate and key for upsd

`openssl` (the program) should be in your PATH, unless you installed it from source yourself, in which case it may be in `/usr/local/ssl/bin`.

Use the following command to create the certificate:

```
openssl req -new -x509 -nodes -out upsd.crt -keyout upsd.key
```

You can also put a `-days nnn` in there to set the expiration. If you skip this, it may default to 30 days. This is probably not what you want.

It will ask several questions. What you put in there doesn't matter a whole lot, since nobody is going to see it for now. Future versions of the clients may present data from it, so you might use this opportunity to identify each server somehow.

Figure out the hash for the key

Use the following command to determine the hash of the certificate:

```
openssl x509 -hash -noout -in upsd.crt
```

You'll get back a single line with 8 hex characters. This is the hash of the certificate, which is used for naming the client-side certificate. For the purposes of this example the hash is **0123abcd**.

Install the client-side certificate

Use the following commands to install the client-side certificate:

```
mkdir <certpath>
chmod 0755 <certpath>
cp upsd.crt <certpath>/<hash>.0
```

Example:

```
mkdir /usr/local/ups/etc/certs
chmod 0755 /usr/local/ups/etc/certs
cp upsd.crt /usr/local/ups/etc/certs/0123abcd.0
```

If you already have a file with that name in there, increment the 0 part until you get a unique filename that works.

If you have multiple client systems (like `upsmon` instances in secondary mode), be sure to install this file on them as well.

We recommend making a directory under your existing `confpath` to keep everything in the same place. Remember the path you created, since you will need to put it in `upsmon.conf` later.

It must not be writable by unprivileged users, since someone could insert a new client certificate and fool `upsmon` into trusting a fake `upsd`.

Create the combined file for upsd

To do so, use the below commands:

```
cat upsd.crt upsd.key > upsd.pem
chown root:nut upsd.pem
chmod 0640 upsd.pem
```

This file must be kept secure, since anyone possessing it could pretend to be `upsd` and harvest authentication data if they get a hold of port 3493.

Having it owned by `root` and readable by group `nut` allows `upsd` to read the file without being able to change the contents. This is done to minimize the impact if someone should break into `upsd`. NUT reads the key and certificate files after dropping privileges and forking.

Note on certification authorities (CAs) and signed keys

There are probably other ways to handle this, involving keys which have been signed by a CA you recognize. Contact your local SSL guru.

Install the server-side certificate

Install the certificate with the following command:

```
mv upsd.pem <upsd certfile path>
```

Example:

```
mv upsd.pem /usr/local/ups/etc/upsd.pem
```

After that, edit your `upsd.conf` and tell it where to find it:

```
CERTFILE /usr/local/ups/etc/upsd.pem
```

Clean up the temporary files

```
rm -f upsd.crt upsd.key
```

Restart upsd

It should come back up without any complaints. If it says something about keys or certificates, then you probably missed a step.

If you run `upsd` as a separate user id (like `nutsrv`), make sure that user can read the `upsd.pem` file.

Point upsmon at the certificates

Edit your `upsmon.conf`, and tell it where the `CERTPATH` is:

```
CERTPATH <path>
```

Example:

```
CERTPATH /usr/local/ups/etc/certs
```

Recommended: make upsmon verify all connections with certificates

Put this in `upsmon.conf`:

```
CERTVERIFY 1
```

Without this, there is no guarantee that the `upsd` is the right host. Enabling this greatly reduces the risk of man in the middle attacks.

This effectively forces the use of SSL, so don't use this unless all of your `upsd` hosts are ready for SSL and have their certificates in order.

Recommended: force upsmon to use SSL

Again in `upsmon.conf`:

```
FORCESSL 1
```

If you don't use `CERTVERIFY 1`, then this will at least make sure that nobody can sniff your sessions without a large effort. Setting this will make `upsmon` drop connections if the remote `upsd` doesn't support SSL, so don't use it unless all of them have it running.

10.6.2 NSS backend usage

This section describes how to enable NUT SSL support using [Mozilla NSS](#).

Install NSS

Install [Mozilla NSS](#) as usual, either from source or binary packages. If using binary packages, be sure to include the developer libraries, and nss-tools (for `certutil`).

Recompile and install NUT

Recompile NUT from source, starting with `configure --with-nss`.

Then install everything as usual.

Create certificate and key for the host

NSS (package generally called `libnss3-tools`) will install a tool called `certutil`. It will be used to generate certificates and manage certificate database.

Certificates should be signed by a certification authorities (CAs). Following commands are typical samples, contact your SSL guru or security officer to follow your company procedures.

GENERATE A SERVER CERTIFICATE FOR `UPSD`:

- Create a directory where store the certificate database: `mkdir cert_db`
- Create the certificate database: `certutil -N -d cert_db`
- Import the CA certificate: `certutil -A -d cert_db -n "My Root CA" -t "TC," -a -i rootca.crt`
- Create a server certificate request (here called "My nut server"): `certutil -R -d cert_db -s "CN=My nut server,O=MyCompany" -a -o server.req`
- Make your CA sign the certificate (produces `server.crt`)
- Import the signed certificate into server database: `certutil -A -d cert_db -n "My nut server" -a -i server.crt -t ","`
- Display the content of certificate server: `certutil -L -d cert_db`

Clients and servers in the same host could share the same certificate to authenticate them or use different ones in same or different databases. The same operation can be done in same or different databases to generate other certificates.

Create a self-signed CA certificate

NSS provides a way to create self-signed certificate which can acting as CA certificate, and to sign other certificates with this CA certificate. This method can be used to provide a CA certification chain without using an "official" certificate authority.

GENERATE A SELF-SIGNED CA CERTIFICATE:

- Create a directory where store the CA certificate database: `mkdir CA_db`
- Create the certificate database: `certutil -N -d CA_db`
- Generate a certificate for CA: `certutil -S -d CA_db -n "My Root CA" -s "CN=My CA,O=MyCompany,ST=MyState" -t "CT," -x -2` (Do not forget to answer Yes to the question "Is this a CA certificate [y/N]?")
- Extract the CA certificate to be able to import it in `upsd` (or `upsmon`) certificate database: `certutil -L -d CA_db -n "My Root CA" -a -o rootca.crt`
- Sign a certificate request with the CA certificate (simulate a real CA signature): `certutil -C -d CA_db -c "My Root CA" -a -i server.req -o server.crt -2 -6`

Install the server-side certificate

Just copy the database directory (just the directory and included 3 database .db files) to the right place, such as `/usr/local/ups/etc`

```
mv cert_db /usr/local/ups/etc/
```

upsd (required): certificate database and self certificate

Edit the `upsd.conf` to tell where find the certificate database:

```
CERTPATH /usr/local/ups/etc/cert_db
```

Also tell which is the certificate to send to clients to authenticate itself and the password to decrypt private key associated to certificate:

```
CERTIDENT "certificate name" "database password"
```

Note

Generally, the certificate name is the server domain name, but is not a hard rule. The certificate can be named as useful.

upsd (optional): client authentication

Note

This functionality is disabled by default. To activate it, recompile NUT with `WITH_CLIENT_CERTIFICATE_VALIDATION` defined:

```
make CFLAGS="-DWITH_CLIENT_CERTIFICATE_VALIDATION"
```

UPSD can accept three levels of client authentication. Just specify it with the directive `CERTREQUEST` with the corresponding value in the `upsd.conf` file:

- **NO**: no client authentication.
- **REQUEST**: a certificate is request to the client but it is not strictly validated. If the client does not send any certificate, the connection is closed.
- **REQUIRE**: a certificate is requested to the client and if it is not valid (no validation chain) the connection is closed.

Like CA certificates, you can add many "trusted" client and CA certificates in server's certificate databases.

upsmon (required): upsd authentication

In order for `upsmon` to securely connect to `upsd`, it must authenticate it. You must associate an `upsd` host name to security rules in `upsmon.conf` with the directive `CERTHOST`.

`CERTHOST` associates a hostname to a certificate name. It also determines whether a SSL connection is mandatory, and if the server certificate must be validated.

```
CERTHOST "hostname" "certificate name" "certverify" "forcessl"
```

If the flag `forcessl` is set to 1, and `upsd` answers that it can not connect with SSL, the connection closes.

If the flag `certverify` is set to 1 and the connection is done in SSL, `upsd`'s certificate is verified and its name must be the specified "certificate name".

To prevent security leaks, you should set all `certverify` and `forcessl` flags to 1 (force SSL connection and validate all certificates for all peers).

You can specify `CERTVERIFY` and `FORCESSL` directive (to 1 or 0) to define a default security rule to apply to all host not specified with a dedicated `CERTHOST` directive.

If a host is not specified in a `CERTHOST` directive, its expected certificate name is its hostname.

upsmon (optional): certificate database and self certificate

Like `upsd`, `upsmon` may need to authenticate itself (`upsd`'s `CERTREQUEST` directive set to `REQUEST` or `REQUIRE`).

It must access to a certificate (and its private key) in a certificate database configuring `CERTPATH` and `CERTIDENT` in `upsmon.conf` in the same way as `upsd`.

```
CERTPATH /usr/local/ups/etc/cert_db
CERTIDENT "certificate name" "database password"
```

10.6.3 Restart upsd

It should come back up without any complaints. If it says something about keys or certificates, then you probably missed a step.

If you run `upsd` as a separate user ID (like `nutsrv`), make sure that user can read files in the certificate directory. NUT reads the keys and certificates after forking and dropping privileges.

10.6.4 Restart upsmon

You should see something like this in the syslog from `upsd`:

```
foo upsd[1234]: Client mon@localhost logged in to UPS [myups] (SSL)
```

If `upsd` or `upsmon` give any error messages, or the (SSL) is missing, then something isn't right.

If in doubt about `upsmon`, start it with `-D` so it will stay in the foreground and print debug messages. It should print something like this every couple of seconds:

```
polling ups: myups@localhost [SSL]
```

Obviously, if the [SSL] isn't there, something's broken.

10.6.5 Recommended: sniff the connection to see it for yourself

Using `tcpdump`, `Wireshark` (Ethereal), or another network sniffer tool, tell it to monitor port 3493/tcp and see what happens. You should only see `STARTTLS` go out, OK `STARTTLS` come back, and the rest will be certificate data and then seemingly random characters.

If you see any plaintext besides that (USERNAME, PASSWORD, etc.) then something is not working.

10.6.6 Potential problems

If you specify a certificate expiration date, you will eventually see things like this in your syslog:

```
Oct 29 07:27:25 rktoy upsmon[3789]: Poll UPS [for750@rktoy] failed -  
SSL error: error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE: certificate  
verify failed
```

You can verify that it is expired by using openssl to display the date:

```
openssl x509 -enddate -noout -in <certfile>
```

It'll display a date like this:

```
notAfter=Oct 28 20:05:32 2002 GMT
```

If that's after the current date, you need to generate another cert/key pair using the procedure above.

10.6.7 Conclusion

SSL support should be considered stable but purposely under-documented since various bits of the implementation or configuration may change in the future. In other words, if you use this and it stops working after an upgrade, come back to this file to find out what changed.

This is why the other documentation doesn't mention any of these directives yet. SSL support is a treat for those of you that RTFM.

There are also potential licensing issues for people who ship binary packages since NUT is GPL and OpenSSL is not compatible with it. You can still build and use it yourself, but you can't distribute the results of it. Or maybe you can. It depends on what you consider "essential system software", and some other legal junk that we're not going to touch.

Other packages have solved this by explicitly stating that an exception has been granted. That is (purposely) impossible here, since NUT is the combined effort of many people, and all of them would have to agree to a license change. This is actually a feature, since it means nobody can unilaterally run off with the source — not even the NUT team.

Note that the replacement of OpenSSL by Mozilla Network Security Services (NSS) should avoid the above licensing issues.

10.7 chrooting and other forms of paranoia

It has been possible to run the drivers and upsd in a chrooted jail for some time, but it involved a number of evil hacks. From the 1.3 series, a much saner chroot behavior exists, using BIND 9 as an inspiration.

The old way involved creating an entire tree, complete with libraries, a shell (!), and many auxiliary files. This was hard to maintain and could have become an interesting playground for an intruder. The new way is minimal, and leaves little in the way of usable materials within the jail.

This document assumes that you already have created at least one user account for the software to use. If you're still letting it fall back on "nobody", stop right here and go figure that out first. It also assumes that you have everything else configured and running happily all by itself.

10.7.1 Generalities

Essentially, you need to create your configuration directory and state path in their own little world, plus a special device or two.

For the purposes of this example, the chroot jail is `/chroot/nut`. The programs have been built with the default prefix, so they are using `/usr/local/ups`. First, create the `confpath` and bring over a few files.

```
mkdir -p /chroot/nut/usr/local/ups/etc
cd /chroot/nut/usr/local/ups/etc
cp -a /usr/local/ups/etc/upsd.users .
cp -a /usr/local/ups/etc/upsd.conf .
cp -a /usr/local/ups/etc/ups.conf .
```

We're using `cp -a` to maintain the permissions on those files.

Now bring over your state path, maintaining the same permissions as before.

```
mkdir -p /chroot/nut/var/state
cp -a /var/state/ups /chroot/nut/var/state
```

Next we must put `/etc/localtime` inside the jail, or you may get very strange readings in your syslog. You'll know you have this problem if `upsd` shows up as UTC in the syslog while the rest of the system doesn't.

```
mkdir -p /chroot/nut/etc
cp /etc/localtime /chroot/nut/etc
```

Note that this is not `cp -a`, since we want to copy the **content**, not the symlink that it may be on some systems.

Finally, create a tiny bit of `/dev` so the programs can enter the background properly — they redirect file descriptors into the bit bucket to make sure nothing else grabs fds 0-2.

```
mkdir -p /chroot/nut/dev
cp -a /dev/null /chroot/nut/dev
```

Try to start your driver(s) and make sure everything fires up as before.

```
upsdrvctl -r /chroot/nut -u nutdev start
```

Once your drivers are running properly, try starting `upsd`.

```
upsd -r /chroot/nut -u nutsrv
```

Check your syslog. If nothing is complaining, try running clients like `upsc` and `upsmon`. If they seem happy, then you're done.

10.7.2 symlinks

After you do this, you will have two copies of many things, like the `confpath` and the `state path`. I recommend deleting the "real" `/var/state/ups`, replacing it with a symlink to `/chroot/nut/var/state/ups`. That will let other programs reference the `.pid` files without a lot of hassle.

You can also do this with your `confpath` and point `/usr/local/ups/etc` (or equivalent on your system) at `/chroot/nut/usr/local/ups/etc` unless you're worried about something hurting the files inside that directory. In that case, you should maintain a "golden" copy and push it into the `chroot` path after making changes.

The `upsdrvctl` itself does not `chroot`, so the `ups.conf` still needs to be in the usual `confpath`.

10.7.3 upsmon

This has not yet been applied to `upsmon`, since it can be quite complicated when there are notifiers that need to be run. One possibility would be for `upsmon` to have three instances:

- privileged root parent that listens for a shutdown command
- unprivileged child that listens for notify events
- unprivileged `chrooted` child that does network I/O

This one is messy, and may not happen for some time, if ever.

10.7.4 Config files

You may now set `chroot=` and `user=` in the global section of `ups.conf`.

The `upsd` chroots before opening any config files, so there is no way to add support for that in `upsd.conf` at the present time.

A Glossary

This section document the various acronyms used throughout the present documentation.

ATS

Automatic Transfer Switch.

NUT

Network UPS Tools.

PDU

Power Distribution Unit.

PSU

Power Supply Units.

SCD

Solar Controller Device.

UPS

Uninterruptible Power Supply.

B Acknowledgements / Contributions

This project is the result of years of work by many individuals and companies.

Many people have written or tweaked the software; the drivers, clients, server and documentation have all received valuable attention from numerous sources.

Many of them are listed within the source code, `AUTHORS` file, release notes, and mailing list archives, but some prefer to be anonymous.

Companies and organizations that have helped with various aspects of project infrastructure are listed in [Acknowledgements of organizations which help with NUT CI and other daily operations](#) (or `README.adoc` in NUT sources for up-to-date information).

This software would not be possible without their help.

B.1 The NUT Team

B.1.1 Active members

- Jim Klimov: project leader (since 2020), OpenIndiana and OmniOS packager, CI dev/ops and infra
 - Arnaud Quette: ex-project leader (from 2005 to 2020), Debian packager and jack of all trades
 - Charles Lepple: senior lieutenant
 - Daniele Pezzini: senior developer
 - Kjell Claesson: senior developer
-

- Alexander Gordeev: junior developer
- Michal Soltys: junior developer
- David Goncalves: Python developer
- Jean Perriault: web consultant
- Eric S. Raymond: Documentation consultant
- Roger Price: Documentation specialist
- Oden Eriksson: Mandriva packager
- Stanislav Brabec: Novell / SUSE packager
- Michal Hlavinka: Redhat packager
- Antoine Colombier: trainee

For an up to date list of NUT developers, refer to [GitHub](#).

B.1.2 Retired members

- Russell Kroll: Founder, and project leader from 1996 to 2005
 - Arjen de Korte: senior lieutenant
 - Peter Selinger: senior lieutenant
 - Carlos Rodrigues: author of the "megatec" drivers, removing the numerous drivers for Megatec / Q1 protocol. These drivers have now been replaced by blazer_ser and blazer_usb
 - Niels Baggesen: ported and heavily extended upscore2 to NUT 2.0 driver model
 - Niklas Edmundsson: has worked on 3-phase support, and upscore2 updates
 - Martin Loyer: has worked a bit on mge-utalk
 - Jonathan Dion: MGE internship (summer 2006), who has worked on configuration
 - Doug Reynolds: has worked on CyberPower support (powerpanel driver)
 - Jon Gough: has worked on porting the megatec driver to USB (megatec_usb)
 - Dominique Lallement: Consultant (chairman of the USB/HID PDC Forum)
 - Julius Malkiewicz: junior developer
 - Tomas Smetana: former Redhat packager (2007-2008)
 - Frederic Bohe: senior developer, Eaton contractor (2009-2013)
 - Emilien Kia: senior developer
 - Václav Krpec: junior developer
-

B.2 Supporting manufacturers

B.2.1 UPS manufacturers

- **Eaton**, has been the main NUT supporter in the past, between 2007 and 2011, continuing MGE UPS SYSTEMS efforts. In 2022, the Eaton NUT-based companion software bundle sources developed (pre-?)2013-2018 were contributed and re-licensed to become part of NUT. As such, Eaton has been:
 - providing extensive technical documents (Eaton protocols library),
 - providing units to developers of NUT and related projects,
 - hosting the networkupstools.org webserver (from 2007 to August 2012),
 - providing artwork,
 - promoting NUT in general,
 - supporting its customers using NUT.



Warning

The situation has evolved, and since 2011 Eaton does not support NUT anymore. This may still evolve in the future.

But for now, please do not consider anymore that buying Eaton products will provide you with official support from Eaton, or a better level of device support in NUT.

- **AMETEK Powervar**, through Andrew McCartney, has added support for all AMETEK Powervar UPM models as usb-hid UPS.
- **Gamatronic**, through Nadav Moskovitch, has revived the *sec* driver (as gamatronic), and expanded a bit genericups for its UPSs with alarm interface.
- **EVER Power Systems** added a USB HID subdriver for EVER UPSes (Sinline RT Series, Sinline RT XL Series, ECO PRO AVR CDS Series).
- **Microdowell**, through Elio Corbolante, has created the *microdowell* driver to support the Enterprise Nxx/Bxx serial devices. The company also proposes NUT as an alternative to its software for **Linux / Unix**.
- **Powercom**, through Alexey Morozov, has provided **extensive information** on its USB/HID devices, along with development units.

They also have a **page suggesting NUT** as the software to support a wide range of their USB HID capable models.
- **Riello UPS**, through Massimo Zampieri, has provided **all protocols information**. Elio Parisi has also created *riello_ser* and *riello_usb* to support these protocols.
- **Tripp Lite**, through Eric Cobb, has provided test results from connecting their HID-compliant UPS hardware to NUT. Some of this information has been incorporated into the NUT hardware compatibility list, and the rest of the information is available via the **list archives**.
- **NAG**, through Alexey Kazancev and Igor Ryabov, has added support for SNR-UPS-LID-XXXX models as usb-hid UPS.
- **Ablere Electronics Co., Ltd.** contributed the *ablerex* subdriver for *blazer_usb*, handling Ablere MP, ARES Plus, MSII MSIII, GRs and GRs Plus models.

B.2.2 Appliances manufacturers

- **OpenGear** has worked with NUT's leader to successfully develop and integrate PDU support. Opendgear, through Scott Burns, and Robert Waldie, has submitted several patches.

B.3 Other contributors

- Pavel Korensky's original apcd provided the inspiration for pursuing APC's smart protocol in 1996
- Eric Lawson provided scans of the OneAC protocol
- John Marley used OCR software to transform the SEC protocol scans into a HTML document
- Chris McKinnon scanned and converted the Fortress protocol documentation
- Tank provided documentation on the Belkin/Delta protocol
- Potrans provided a Fenton PowerPal 600 (P series) for development of the safenet driver.

B.4 Older entries (before 2005)

- MGE UPS SYSTEMS was the previous NUT sponsor, from 2002 until its partial acquisition by Eaton. They provided protocols information, many units for development of NUT-related projects. Several drivers such as mge-utalk, mge-shut, snmp-ups, hidups, and usbhid-ups are the result of this collaboration, in addition to the WMNut, MGE HID Parser the libhid projects, ... through Arnaud Quette (who was also an MGE employee). All the MGE supporters have gone with Eaton (through MGE Office Protection Systems), which was temporarily the new NUT sponsor.
- Fenton Technologies contributed a PowerPal 660 to the project. Their open stance and quick responses to technical inquiries were appreciated for making the development of the fentonups driver possible. Fenton has since been acquired by **Metapo**.
- Bo Kersey of **VirCIO** provided a Best Power Fortress 750 to facilitate the bestups driver.
- Invensys Energy Systems provided the SOLA/Best "Phoenixtec" protocol document. SOLA has since been acquired by Eaton.
- PowerKinetics technical support provided documentation on their MiniCOL protocol, which is archived in the NUT protocol library. PowerKinetics was acquired by the **JST Group** in June 2003.
- **Cyber Power Systems** contributed a 700AVR model for testing and development of the cyberpower driver.
- **Liebert Corporation** supplied serial test boxes and a UPStation GXT2 with the Web/SNMP card for development of the liebert driver and expansion of the existing snmp-ups driver. Liebert has since been acquired by **Emerson**.

Note

If a company or individual isn't listed here, then we probably don't have enough information about the situation. Developers are kindly requested to report vendor contributions to the NUT team so this list may reflect their help, as well as convey a sense of official support (at least, that drivers were proposed according to the know-how coming from specs and knowledge about hardware and firmware capabilities, and not acquired via reverse engineering with a certain degree of unreliability and incompleteness). If we have left you out, please send us some mail or post a pull request to update this document in GitHub.

C NUT command and variable naming scheme

RFC 9271 Recording Document

This document is defined by RFC 9271 published by IETF at <https://www.rfc-editor.org/info/rfc9271> and is referenced as the document of record for the variable names and the instant commands used in the protocol described by the RFC.

On behalf of the RFC, this document records the names of variables describing the abstracted state of an UPS or similar power distribution device, and the instant commands sent to the UPS using command `INSTCMD`, as used in commands and messages between the Attachment Daemon (the `upsd` in case of NUT implementation of the standard) and the clients.

This document defines the standard names of NUT commands and variables (not to be confused with [device status data](#) described in the `docs/new-drivers.txt` in NUT source codebase).

Developers should use the names recorded here, with `dstate` functions and data mappings provided in NUT drivers for interactions with power devices.

If you need to express a state which cannot be described by any existing name, please make a request to the NUT developers' mailing list for definition and assignment of a new name. Clients using unrecorded names risk breaking at a future update. If you wish to experiment with new concepts before obtaining your requested variable name, you should use a name of the form `experimental.x.y` for those states.

Put another way: if you make up a name that is not in this list and it gets into the source code tree, and then the NUT community comes up with a better name later, clients that already use the undocumented variable will break when it is eventually changed. An explicitly "experimental" data point is less surprising in this regard.

Similarly, some source files (`drivers/*-mib.c` and `drivers/*-hid.c`) may mention data point names following the pattern of `unmapped.x.y`. These are generated by helper scripts which walk the reports from SNMP and USB HID devices, respectively `scripts/subdriver/gen-snm-subdriver.sh` and `scripts/subdriver/gen-usbhid-subdriver.sh` and assign names based on strings in those reports. The unmapped entries should not be exposed in "production" builds of the NUT drivers. They are an aid for developers to know that such entries are served by their device, so an existing standard NUT name can be assigned for the concept (or new name negotiated with the community), but are normally hidden with `#if WITH_UNMAPPED_DATA_POINTS` clauses which can be enabled in custom NUT builds by use of `./configure --with-unmapped-data-points` option.

Note

In the descriptions, "opaque" means programs should not attempt to parse the value for that variable as it may vary greatly from one UPS (or similar device) to the next. These strings are best handled directly by the user.

C.1 Structured naming

All standard NUT names of variables and commands are structured, with a certain domain-specific prefix and purpose-specific suffix parts. NUT tools provide and interpret them as dot-separated strings (although third-party tools might restructure them by cutting and pasting at the dot separation location, e.g. to represent as a JSON data tree or as data model classes for specific programming languages).

If you would be making a parser of this information, please do also note that in some **but not all** cases there is a defined data point for some reading or command at the "root level" of what evolved to be a collection of further structured related information (and there are no guarantees for future evolution in this regard), for example:

- an `input.voltage` reports the momentary voltage level value and there is a `input.voltage.maximum` for a certain related detail;
- conversely, there are several items like `input.transfer.reason` but there is no actual `input.transfer` report.

There may be more layers than two (e.g. `input.voltage.low.warning`), and in certain cases detailed below there may be a variable component in the practical values (e.g. the `n` in `ambient.n.temperature.alarm` variable or `outlet.n.load.off` command names).

C.2 Numeric format

Depending on the use-case, decimal numbers may be represented as integers or as floating-point numbers with a dot character as the separator for the fractional part (typically one or two digits long). Leading zeroes may be present. Leading (negative) sign characters are possible, although use-cases for them are rare (if any).

Spaces or commas must not be used inside the numeric values.

Scientific notation (with mantissa/exponent) must not be used to represent numeric values set into variables, to serve the values as exact as we have them and keep the client-side parsing simple and predictable.

For example: "01200.2" and "1200.20" are valid, while "1,200.20" and "1200,20" and "1 200.20" and "1.2e4" are invalid.

Programming note: floating-point numbers should be emitted using the `%f` format specifier in the `C printf` family of methods and derived methods (including `NUT dstate_setinfo()` in the driver code). Specifiers like `%e` and `%g` which can emit the scientific notation should be avoided when setting variable values (directly in code or when providing format string patterns in mapping tables). They may however be used in debug traces, where reasonable.

Note that in some cases (e.g. USB vendor and product identifiers) technically numeric values may be reported as hexadecimal and should be treated generally as opaque strings (with the consumer ascribing a known meaning to certain variable names).

C.3 Time and Date format

When possible, dates should be expressed in ISO 8601 and RFC 3339 compatible Calendar format, that is to say "YYYY-MM-DD", or otherwise a Combined Date and Time representation (`<date>T<time>`, so "YYYY-MM-DDThh:mm"). Separators for the date (hyphen) and time (colon) components are required to conform to both ISO 8601 "extended" format and RFC 3339 required format.

In the case of Date and Time representation, a timezone can be added as per RFC 3339 and the newer revisions of the ISO 8601 standard (which allow for negative offsets):

- by appending the letter `Z` for UTC (e.g. "YYYY-MM-DDThh:mmZ"), or
- by appending the complete "hours:minutes" positive or negative time offsets from UTC (e.g. "YYYY-MM-DDThh:mm+03:00").

For more details see examples at [Wikipedia page on ISO 8601](#) and the publicly available RFC at [RFC 3339](#).

Other representations from those specifications are not necessarily supported.

Note

Values of certain variables may be propagated from device reports formally as opaque strings, which happen to convey a date/time value (commonly the device or battery manufacture date, replacement date, last self-test or calibration time stamp, device clock, etc.) in some format, not necessarily a standard one.

While the drivers may convert them from original vendor-provided markup to the standard Time and Date format described above (if the formula is known for certain — e.g. which locale is used by the device, which part of that string is the year/month/-day, or how to add offset or prefix for the year, etc.), they are not generally required to do so.

C.4 Variables

C.4.1 device: General unit information

Note

Some of these data will be redundant with `ups.*` information during a transition period. The `ups.*` data will then be removed.

Name	Description	Example value
device.model	Device model	BladeUPS
device.mfr	Device manufacturer	Eaton
device.serial	Device serial number (opaque string)	WS9643050926
device.type	Device type (ups, pdu, scd, psu, ats)	ups
device.description	Device description (opaque string)	Some ups
device.contact	Device administrator name (opaque string)	John Doe

Name	Description	Example value
device.location	Device physical location (opaque string)	1st floor
device.part	Device part number (opaque string)	123456789
device.macaddr	Physical network address of the device	68:b5:99:f5:89:27
device.uptime	Device uptime in seconds	1782
device.count	Total number of daisy chained devices	1
device.usb.version	Device (firmware-reported) USB version	01.29

Note

When present, `device.count` implies daisychain support. For more information, refer to the [NUT daisychain support notes](#) chapter of the user manual and developer guide.

C.4.2 ups: General unit information

Name	Description	Example value
ups.status	UPS status (opaque string comprised of space-separated tokens; many of those are ascribed certain meanings)	OL
ups.alarm	UPS alarms (opaque string, may be a collection of whole sentences; separate entries may be enclosed in brackets for convenience, but this standard does not require it)	OVERHEAT [EEPROM Error]
ups.time	Internal UPS clock time (opaque string)	12:34
ups.date	Internal UPS clock date (opaque string)	01-02-03
ups.model	UPS model	SMART-UPS 700
ups.mfr	UPS manufacturer	APC
ups.mfr.date	UPS manufacturing date (opaque string)	10/17/96
ups.serial	UPS serial number (opaque string)	WS9643050926
ups.vendorid	Vendor ID for USB devices	0463
ups.productid	Product ID for USB devices	0001
ups.firmware	UPS firmware (opaque string)	50.9.D
ups.firmware.aux	Auxiliary device firmware	4Kx
ups.temperature	UPS temperature (degrees C)	042.7
ups.load	Load on UPS (percent)	023.4
ups.load.high	Load when UPS switches to overload condition ("OVER") (percent)	100
ups.id	UPS system identifier (opaque string)	Sierra
ups.delay.start	Interval to wait before restarting the load (seconds)	0
ups.delay.reboot	Interval to wait before rebooting the UPS (seconds)	60
ups.delay.shutdown	Interval to wait after shutdown with delay command (seconds)	20
ups.timer.start	Time before the load will be started (seconds)	30
ups.timer.reboot	Time before the load will be rebooted (seconds)	10

Name	Description	Example value
ups.timer.shutdown	Time before the load will be shutdown (seconds)	20
ups.test.interval	Interval between self tests (seconds)	1209600 (two weeks)
ups.test.result	Results of last self test (opaque string)	Bad battery pack
ups.test.date	Date of last self test (opaque string)	07/17/12
ups.display.language	Language to use on front panel (* opaque)	E
ups.contacts	UPS external contact sensors (* opaque)	F0
ups.efficiency	Efficiency of the UPS (ratio of the output current on the input current) (percent)	95
ups.power	Current value of apparent power (Volt-Amps)	500
ups.power.nominal	Nominal value of apparent power (Volt-Amps)	500
ups.realpower	Current value of real power (Watts)	300
ups.realpower.nominal	Nominal value of real power (Watts)	300
ups.beeper.status	UPS beeper status (enabled, disabled or muted)	enabled
ups.type	UPS type (* opaque)	offline
ups.mode	Current UPS mode (see the note below)	line-interactive
experimental.ups.mode.buzzwords	UPS mode details, not classified (opaque string, presumably comprised of space-separated tokens; see the note below)	vendor:eaton:ECO
ups.watchdog.status	UPS watchdog status (enabled or disabled)	disabled
ups.start.auto	UPS starts when mains is (re)applied	yes
ups.start.battery	Allow to start UPS from battery	yes
ups.start.reboot	UPS coldstarts from battery (enabled or disabled)	yes
ups.shutdown	Enable or disable UPS shutdown ability (poweroff)	enabled

Note

When present, the value of `ups.start.auto` has an impact on `shutdown.*` commands. For the sake of coherence, shutdown commands will set `ups.start.auto` to the right value before issuing the command. That is, `shutdown.stayoff` will first set `ups.start.auto` to `no`, while `shutdown.return` will set it to `yes`.

Note

When present, the value of `ups.mode` specifies the currently enabled mode of operation of the inverter and other components in the UPS. Some devices are wired to only have one mode, others may support several modes and usually have a way to select which one you want to be currently active, either via protocol commands or by a physical switch.

There are many marketing keywords of different vendors and device generations that sometimes correspond to same or very similar concepts, other times overlap with wildly different meanings.

For example, some devices may be "online" (doing double-conversion and feeding the load from battery even when wall power is available) when charging or compensating for poor quality of input power, but become "line-interactive" or even power off some of their electronics when the input is deemed reliable.

The `ups.mode` can have one of the following standard values, possibly changing over time or never changing (for devices with one known mode):

- `online`: battery is always charging on one side, and always feeds the inverter and so the load on the other (pros: instant protection from outages; cons: higher overheads of the UPS itself, possibly faster wear of the battery);
- `line-interactive`: battery is charged and then kept at rest, load is fed from input, but in case of outage or other troubles the inverter or other compensation mechanism (trim/buck) would be fired up (after a small but non-trivial delay);
- `bypass`: inverter and battery are bypassed (e.g. for maintenance), so input directly feeds the output, until it suddenly does not.

The `experimental.ups.mode.buzzwords` can have one or more values, separated by spaces, to provide information we know from the device but have not yet agreed how to reflect it in a well-structured fashion (hence `experimental` namespace is used):

- `vendor:VENDORNAME:MODENAME`: we know that the "vendor's marketing buzzword" mode is activated. Users may read vendor documentation for their device model, and know better than the driver what this actually means for them. It is recommended to keep vendor names lower-cased and mode names upper-cased, for more deterministic matching and sorting in NUT clients.
 - A number of `MODENAME` values/patterns may be interpreted by `upsmmon(8)` to produce notifications about entering or exiting the "ECO" mode state.

Other values may be introduced later.

See also `output.inverter.latency`.

Note

When possible, time-stamps and dates should be expressed as detailed above in the Time and Date format chapter.

C.4.3 input: Incoming line/power information

Name	Description	Example value
<code>input.voltage</code>	Input voltage (V)	121.5
<code>input.voltage.maximum</code>	Maximum incoming voltage seen (V)	130
<code>input.voltage.minimum</code>	Minimum incoming voltage seen (V)	100
<code>input.voltage.status</code>	Status relative to the thresholds	critical-low
<code>input.voltage.low.warning</code>	Low warning threshold (V)	205
<code>input.voltage.low.critical</code>	Low critical threshold (V)	200
<code>input.voltage.high.warning</code>	High warning threshold (V)	230
<code>input.voltage.high.critical</code>	High critical threshold (V)	240
<code>input.voltage.nominal</code>	Nominal input voltage (V)	120
<code>input.voltage.extended</code>	Extended input voltage range	no
<code>input.transfer.delay</code>	Delay before transfer to mains (seconds)	60

Name	Description	Example value
input.transfer.reason	Reason for last transfer to battery (* opaque)	T
input.transfer.low	Low voltage transfer point (V)	91
input.transfer.high	High voltage transfer point (V)	132
input.transfer.low.min	smallest settable low voltage transfer point (V)	85
input.transfer.low.max	greatest settable low voltage transfer point (V)	95
input.transfer.high.min	smallest settable high voltage transfer point (V)	131
input.transfer.high.max	greatest settable high voltage transfer point (V)	136
input.eco.switchable	Input High Efficiency (aka ECO) mode switch (opaque string)	normal
input.sensitivity	Input power sensitivity	H (high)
input.quality	Input power quality (* opaque)	FF
input.current	Input current (A)	4.25
input.current.nominal	Nominal input current (A)	5.0
input.current.status	Status relative to the thresholds	critical-high
input.current.low.warning	Low warning threshold (A)	4
input.current.low.critical	Low critical threshold (A)	2
input.current.high.warning	High warning threshold (A)	10
input.current.high.critical	High critical threshold (A)	12
input.feed.color	Color of the input feed (opaque string)	3831236
input.feed.desc	Description of the input feed	Feed A
input.frequency	Input line frequency (Hz)	60.00
input.frequency.nominal	Nominal input line frequency (Hz)	60
input.frequency.status	Frequency status	out-of-range
input.frequency.low	Input line frequency low (Hz)	47
input.frequency.high	Input line frequency high (Hz)	63
input.frequency.extended	Extended input frequency range	no
input.transfer.boost.low	Low voltage boosting transfer point (V)	190
input.transfer.boost.high	High voltage boosting transfer point (V)	210
input.transfer.trim.low	Low voltage trimming transfer point (V)	230
input.transfer.trim.high	High voltage trimming transfer point (V)	240
input.transfer.eco.low	Low voltage ECO transfer point (V)	218
input.transfer.bypass.low	Low voltage Bypass transfer point (V)	184
input.transfer.eco.high	High voltage ECO transfer point (V)	241
input.transfer.bypass.high	High voltage Bypass transfer point (V)	264
input.transfer.frequency.bypass.range	Frequency range Bypass transfer point (percent of nominal Hz)	10
input.transfer.frequency.eco.range	Frequency range ECO transfer point (percent of nominal Hz)	5
input.transfer.hysteresis	Threshold of switching protection modes, voltage transfer point (V)	10
input.transfer.bypass.forced	Rule for allow auto Bypass switch (on/off) transfer modes (enabled or disabled)	enabled
input.transfer.bypass.overload	Rule for auto transfer on Bypass when overload (enabled or disabled)	enabled
input.transfer.bypass.outlimits	Rule for auto transfer on Bypass when out of tolerance (enabled or disabled)	enabled

Name	Description	Example value
input.bypass.switchable	Input auto transfer on Bypass when overload or out of tolerance (enabled or disabled)	enabled
input.bypass.switch.on	Automatically put the UPS in Bypass mode	on
input.bypass.switch.off	Automatically take the UPS out of Bypass mode	disabled
input.bypass.voltage	Input bypass voltage (V)	233
input.bypass.frequency	Input bypass frequency (Hz)	50
input.load	Load on (ePDU) input (percent of full)	25
input.realpower	Current sum value of all (ePDU) phases real power (W)	300
input.realpower.nominal	Nominal sum value of all (ePDU) phases real power (W)	850
input.power	Current sum value of all (ePDU) phases apparent power (VA)	500
input.source	The current input power source	1
input.source.preferred	The preferred power source	1
input.phase.shift	Voltage dephasing between input sources (degrees)	181

Input Voltage Hysteresis

The input voltage hysteresis concept refers to a specific behavior related to how some UPS models can handle changes in input voltage.

When the UPS is running normally (powered by utility or generator), it maintains a steady output voltage for your critical equipment. But what if the input voltage "wiggles" a bit due to fluctuations or other minor disturbances?

Rapid switching between UPS protection modes (utility power to battery and vice versa) can stress both the UPS and its connected devices.

So, some UPS models set up thresholds: If the input voltage drops below a certain "Low" level, the UPS won't immediately switch to battery mode. Instead, it waits until it is sure the voltage stays consistently low for a bit. Similarly, if the input voltage rises above another threshold (the "High" level), the UPS won't rush back to normal mode. It waits for stability.

By introducing hysteresis, such an UPS avoids unnecessary toggling, ensuring smoother transitions and better protection for your sensitive and expensive gear.

C.4.4 output: Outgoing power/inverter information

Name	Description	Example value
output.voltage	Output voltage (V)	120.9
output.voltage.nominal	Nominal output voltage (V)	120
output.frequency	Output frequency (Hz)	59.9
output.frequency.nominal	Nominal output frequency (Hz)	60
output.current	Output current (A)	4.25
output.current.nominal	Nominal output current (A)	5.0
output.inverter.latency	Delay of inverter activation when switching to battery (seconds, floating-point)	0.01

Note

One practical aspect that the users may be actually interested in is the `output.inverter.latency`, representing the time gap when an outage begins, after the mains power has disappeared and before the inverter begins to feed the load from battery. Typical values are 0 for double-conversion devices, which always feed the load from battery, and 10msec (0.01 seconds in standard units) for "line-interactive" devices which monitor input status and take time to react to an outage or breach of thresholds. While common computer power sources include elements that allow them to slide over such a short outage, other protected devices (laser printers, Hi-Fi audio) might not, and would restart.

See also `ups.mode` and `experimental.ups.mode.buzzwords`.

C.4.5 Three-phase additions

The additions for three-phase measurements would produce a very long table due to all the combinations that are possible, so these additions are broken down to their base components.

Phase Count Determination

`input.phases` (3 for three-phase, absent or 1 for 1phase)

`output.phases` (as for `input.phases`)

DOMAINs

Any input or output is considered a valid DOMAIN.

- input (should really be called `input.mains`, but keep this for compat)
 - `input.bypass`
 - `input.servicebypass`
- output (should really be called `output.load`, but keep this for compat)
 - `output.bypass`
 - `output.inverter`
 - `output.servicebypass`

Specification (SPEC)

Voltage, current, frequency, etc are considered to be a specification of the measurement.

With this notation, the old 1phase naming scheme becomes DOMAIN.SPEC

Example: `input.current`

CONTEXT

When in three-phase mode, we need some way to specify the target for most measurements in more detail. We call this the CONTEXT.

With this notation, the naming scheme becomes DOMAIN.CONTEXT.SPEC when in three-phase mode.

Example: `input.L1.current`

Valid CONTEXTs

```
L1-L2  \
L2-L3  \
L3-L1  for voltage measurements
L1-N    /
L2-N    /
L3-N    /
```

```
L1  \
L2  for current and power measurements
L3  /
N   - for current measurement
```

Valid SPECS

Note

For cursory readers—the following couple of tables lists just the short SPEC component of the larger DOMAIN.CONTEXT.SPEC naming scheme for phase-aware values, as discussed in other sections of this chapter just above. These are NOT to be used verbatim as complete data-point names!

Valid with/without context (i.e. per phase or aggregated/averaged)

Name	Description
alarm	Alarms for phases, published in ups.alarm
current	Current (A)
current.maximum	Maximum seen current (A)
current.minimum	Minimum seen current (A)
current.status	Status relative to the thresholds
current.low.warning	Low warning threshold (A)
current.low.critical	Low critical threshold (A)
current.high.warning	High warning threshold (A)
current.high.critical	High critical threshold (A)
current.peak	Peak current
voltage	Voltage (V)
voltage.nominal	Nominal voltage (V)
voltage.maximum	Maximum seen voltage (V)
voltage.minimum	Minimum seen voltage (V)
voltage.status	Status relative to the thresholds
voltage.low.warning	Low warning threshold (V)
voltage.low.critical	Low critical threshold (V)
voltage.high.warning	High warning threshold (V)
voltage.high.critical	High critical threshold (V)
power	Apparent power (VA)
power.maximum	Maximum seen apparent power (VA)
power.minimum	Minimum seen apparent power (VA)
power.percent	Percentage of apparent power related to maximum load
power.maximum.percent	Maximum seen percentage of apparent power
power.minimum.percent	Minimum seen percentage of apparent power
realpower	Real power (W)
powerfactor	Power Factor (dimensionless value between 0.00 and 1.00)
crestfactor	Crest Factor (dimensionless value greater or equal to 1)
load	Load on (ePDU) input

Valid without context (i.e. aggregation of all phases):

Name	Description
frequency	Frequency (Hz)
frequency.nominal	Nominal frequency (Hz)
realpower	Current value of real power (Watts)
power	Current value of apparent power (Volt-Amps)

C.4.6 EXAMPLES

Partial Three phase — Three phase example:

```
input.phases: 3
input.frequency: 50.0
input.L1.current: 133.0
input.bypass.L1-L2.voltage: 398.3
output.phases: 3
output.L1.power: 35700
output.powerfactor: 0.82
```

Partial Three phase — One phase example:

```
input.phases: 3
input.L2.current: 48.2
input.N.current: 3.4
input.L3-L1.voltage: 405.4
input.frequency: 50.1
output.phases: 1
output.current: 244.2
output.voltage: 120
output.frequency.nominal: 60.0
```

C.4.7 battery: Any battery details

Name	Description	Example value
battery.charge	Battery charge (percent)	100.0
battery.charge.approx	Rough approximation of battery charge (opaque, percent)	<85
battery.charge.low	Remaining battery level when UPS switches to LB (percent)	20
battery.charge.restart	Minimum battery level for UPS restart after power-off	20
battery.charge.warning	Battery level when UPS switches to "Warning" state (percent)	50
battery.charger.status	Status of the battery charger (see the note below)	charging
battery.charger.type	Type of battery charger	ABM
battery.voltage	Battery voltage (V)	24.84
battery.voltage.cell.max	Maximum battery voltage seen of the Li-ion cell (V)	3.44
battery.voltage.cell.min	Minimum battery voltage seen of the Li-ion cell (V)	3.41

Name	Description	Example value
battery.voltage.nominal	Nominal battery voltage (V)	024
battery.voltage.low	Minimum battery voltage, that triggers FSD status	21,52
battery.voltage.high	Maximum battery voltage (i.e. battery.charge = 100)	26,9
battery.capacity	Battery capacity (Ah)	7.2
battery.capacity.nominal	Nominal battery capacity (Ah)	8.0
battery.current	Battery current (A)	1.19
battery.current.total	Total battery current (A)	1.19
battery.status	Health status of the battery (opaque string)	ok
battery.temperature	Battery temperature (degrees C)	050.7
battery.temperature.cell.max	Maximum battery temperature seen of the Li-ion cell (degrees C)	25.85
battery.temperature.cell.min	Minimum battery temperature seen of the Li-ion cell (degrees C)	24.85
battery.runtime	Battery runtime (seconds)	1080
battery.runtime.low	Remaining battery runtime when UPS switches to LB (seconds)	180
battery.runtime.restart	Minimum battery runtime for UPS restart after power-off (seconds)	120
battery.alarm.threshold	Battery alarm threshold	0 (immediate)
battery.date	Battery installation or last change date (opaque string)	11/14/20
battery.date.maintenance	Battery next change or maintenance date (opaque string)	11/13/24
battery.mfr.date	Battery manufacturing date (opaque string)	2005/04/02
battery.packs	Number of internal battery packs	1
battery.packs.bad	Number of bad battery packs	0
battery.packs.external	Number of external battery packs	1
battery.type	Battery chemistry (opaque string)	PbAc
battery.protection	Prevent deep discharge of battery	yes
battery.energysave	Switch off when running on battery and no/low load	no
battery.energysave.load	Switch off UPS if on battery and load level lower (percent)	5
battery.energysave.delay	Delay before switch off UPS if on battery and load level low (min)	3
battery.energysave.realpower	Switch off UPS if on battery and load level lower (Watts)	10

Note

`battery.charger.status` replaces the historic flags `CHRG` and `DISCHRG` that were exposed through `ups.status`. The `battery.charger.status` can have one of the following values:

- `charging`: battery is charging,
 - `discharging`: battery is discharging,
 - `floating`: battery has completed its charge cycle, and waiting to go to resting mode,
 - `resting`: the battery is fully charged, and not charging nor discharging.
-

Note

When possible, time-stamps and dates should be expressed as detailed above in the Time and Date format chapter.

C.4.8 ambient: Conditions from external probe equipment**Note**

n stands for the sensors index. A special case is "ambient.0" which is equivalent to "ambient" (without index), and represents the default sensor of the device. This is not to be confused with the device embedded sensor, which is published as *ups.temperature*. The most important data is "ambient.count", used to iterate over the whole set of outlets. For more information, refer to the NUT sensors management notes chapter of the user manual.

Name	Description	Example value
ambient.count	Total number of sensors	2
ambient.n.name	Ambient sensor name	sensor 1
ambient.n.id	Ambient sensor identifier (opaque string)	80f09325-2838-5637-b62a-cef9cbe2747
ambient.n.address	Ambient sensor address (opaque string)	1
ambient.n.parent.serial	Ambient sensor parent serial number (opaque string)	U603E34000
ambient.n.mfr	Ambient sensor manufacturer	EATON
ambient.n.model	Ambient sensor model	EMPDT1H1C2
ambient.n.firmware	Ambient sensor firmware	01.03.0011
ambient.n.present	Ambient sensor presence	yes
ambient.n.temperature	Ambient temperature (degrees C)	25.40
ambient.n.temperature.alarm	Temperature alarm (enabled/disabled)	enabled
ambient.n.temperature.status	Ambient temperature status relative to the thresholds	warning-low
ambient.n.temperature.high	Temperature threshold high (degrees C)	60
ambient.n.temperature.high.warning	Temperature threshold high warning (degrees C)	40
ambient.n.temperature.high.critical	Temperature threshold high critical (degrees C)	60
ambient.n.temperature.low	Temperature threshold low (degrees C)	5
ambient.n.temperature.low.warning	Temperature threshold low warning (degrees C)	10
ambient.n.temperature.low.critical	Temperature threshold low critical (degrees C)	5
ambient.n.temperature.maximum	Maximum temperature seen (degrees C)	37.6
ambient.n.temperature.minimum	Minimum temperature seen (degrees C)	18.1
ambient.n.humidity	Ambient relative humidity (percent)	038.8
ambient.n.humidity.alarm	Relative humidity alarm (enabled/disabled)	enabled
ambient.n.humidity.status	Ambient humidity status relative to the thresholds	warning-low
ambient.n.humidity.high	Relative humidity threshold high (percent)	80
ambient.n.humidity.high.warning	Relative humidity threshold high warning (percent)	70

Name	Description	Example value
ambient.n.humidity.high.critical	Relative humidity threshold high critical (percent)	80
ambient.n.humidity.low	Relative humidity threshold low (percent)	10
ambient.n.humidity.low.warning	Relative humidity threshold low warning (percent)	20
ambient.n.humidity.low.critical	Relative humidity threshold low critical (percent)	10
ambient.n.humidity.maximum	Maximum relative humidity seen (percent)	60
ambient.n.humidity.minimum	Minimum relative humidity seen (percent)	13
ambient.n.contacts.x.status	State of the dry contact sensor x	open
ambient.n.contacts.x.config	Configuration of the dry contact sensor x	normal-open
ambient.n.contacts.x.name	Name of the dry contact sensor x	smoke-detector1

NOTE: - ambient.n.contacts.x.status may either be the raw status (*open* or *closed*), or may relate to ambient.n.contacts.x.config. In this case, the value can be *active* or *inactive*.

C.4.9 outlet: Smart outlet management

Note

n stands for the outlet index. A special case is "outlet.0" which is equivalent to "outlet" (without index), and represent the whole set of outlets of the device. The most important data is "outlet.count", used to iterate over the whole set of outlets. For more information, refer to the NUT outlets management and PDU notes chapter of the user manual.

Name	Description	Example value
outlet.count	Total number of outlets	12
outlet.switchable	General outlet switch ability of the unit (yes/no)	yes
outlet.n.id	Outlet system identifier (opaque string)	1
outlet.n.name	Outlet name (opaque string)	A1
outlet.n.desc	Outlet description (opaque string)	Main outlet
outlet.n.groupid	Identifier of the group to which the outlet belongs to	1
outlet.n.switch	Outlet switch control (on/off)	on
outlet.n.status	Outlet switch status (on/off)	on
outlet.n.protect.status	Outlet protection status (opaque string)	protected
outlet.n.alarm	Alarms for outlets and PDU, published in ups.alarm	outlet 1 low voltage warning
outlet.n.switchable	Outlet switch ability (yes/no)	yes
outlet.n.ecocontrol	Master Outlet used to automatically power off the slave outlets	The outlet is not ECO controlled
outlet.n.designator	Outlet designator	AC OUTPUT
outlet.n.autoswitch.charge.low	Remaining battery level to power off this outlet (percent)	80
outlet.n.battery.charge.low	Remaining battery level to power off this outlet (percent)	80

Name	Description	Example value
outlet.n.delay.shutdown	Interval to wait before shutting down this outlet (seconds)	180
outlet.n.delay.start	Interval to wait before restarting this outlet (seconds)	120
outlet.n.timer.shutdown	Time before the outlet load will be shutdown (seconds)	20
outlet.n.timer.start	Time before the outlet load will be started (seconds)	30
outlet.n.current	Current (A)	0.19
outlet.n.current.maximum	Maximum seen current (A)	0.56
outlet.n.current.status	Current status relative to the thresholds	good
outlet.n.current.low.warning	Low warning threshold (A)	0.10
outlet.n.current.low.critical	Low critical threshold (A)	0.05
outlet.n.current.high.warning	High warning threshold (A)	0.30
outlet.n.current.high.critical	High critical threshold (A)	0.40
outlet.n.realpower	Current value of real power (W)	28
outlet.n.voltage	Voltage (V)	247.0
outlet.n.voltage.status	Voltage status relative to the thresholds	good
outlet.n.voltage.low.warning	Low warning threshold (V)	205
outlet.n.voltage.low.critical	Low critical threshold (V)	200
outlet.n.voltage.high.warning	High warning threshold (V)	230
outlet.n.voltage.high.critical	High critical threshold (V)	240
outlet.n.powerfactor	Power Factor (dimensionless, value between 0 and 1)	0.85
outlet.n.crestfactor	Crest Factor (dimensionless, equal to or greater than 1)	1.41
outlet.n.power	Apparent power (VA)	46
outlet.n.type	Physical outlet type	french

outlet.group: groups of smart outlets

This is a refinement of the outlet collection, providing grouped management for a set of outlets. The same principles and data than the outlet collection apply to outlet.group, especially for the indexing *n* and "outlet.group.count".

Most of the data published for outlets also apply to outlet.group, including: id, name (similar as outlet "desc"), color, status, current and voltage (including status, alarm and thresholds). Other actions and settings also apply ({delay,timer}.{shutdown,start})

Some specific data to outlet groups exists:

Name	Description	Example value
outlet.group.n.type	Type of outlet group (OPAQUE)	outlet-section
outlet.group.n.color	Color-coding of the outlets in this group (OPAQUE)	yellow
outlet.group.n.count	Number of outlets in the group	12
outlet.group.n.phase	Electrical phase to which the physical outlet group (Gang) is connected to	L1
outlet.group.n.input	Input to which an outlet group is connected	1

Example:

```
outlet.group.1.current: 0.00
outlet.group.1.current.high.critical: 16.00
outlet.group.1.current.high.warning: 12.80
```

```

outlet.group.1.current.low.warning: 0.00
outlet.group.1.current.nominal: 16.00
outlet.group.1.current.status: good
outlet.group.1.id: 1
outlet.group.1.name: Branch Circuit A
outlet.group.1.phase: L1
outlet.group.1.status: on
outlet.group.1.voltage: 244.23
outlet.group.1.voltage.high.critical: 265.00
outlet.group.1.voltage.high.warning: 255.00
outlet.group.1.voltage.low.critical: 180.00
outlet.group.1.voltage.low.warning: 190.00
outlet.group.1.voltage.status: good
...
outlet.group.count: 3.00

```

C.4.10 driver: Internal driver information

Name	Description	Example value
driver.name	Driver name	usbhid-ups
driver.version	Driver version (NUT release)	X.Y.Z
driver.version.internal	Internal driver version	1.23.45
driver.version.data	Version of the internal data mapping, for generic drivers	Eaton HID 1.31
driver.version.usb	USB library version	libusb-1.0.21
driver.parameter.xxx	Parameter xxx (ups.conf or cmdline -x) setting	(varies)
driver.flag.xxx	Flag xxx (ups.conf or cmdline -x) status	enabled (or absent)
driver.state	Current state in driver's lifecycle, primarily to help readers discern long-running init (with full device walk) or cleanup stages from the stable working loop	init.starting, init.quiet, init.device, init.info, init.updateinfo (first walk), reconnect.trying, reconnect.updateinfo, updateinfo, quiet, dumping, cleanup.upsdrv, cleanup.exit

C.4.11 server: Internal server information

Name	Description	Example value
server.info	Server information	Network UPS Tools upsd vX.Y.Z - https://www.networkupstools.org/
server.version	Server version	X.Y.Z

C.5 Instant commands

Name	Description
load.off	Turn off the load immediately
load.on	Turn on the load immediately
load.off.delay	Turn off the load possibly after a delay
load.on.delay	Turn on the load possibly after a delay

Name	Description
shutdown.default	Run default driver-defined (device-specific) routine, primarily intended for emergency poweroff performed as part of FSD handling; often an alias to other <code>shutdown.*</code> and/or <code>load.off</code> operations or a chain to try several of those. See also <code>sdcommands</code> in common driver options.
shutdown.return	Turn off the load possibly after a delay and return when power is back
shutdown.stayoff	Turn off the load possibly after a delay and remain off even if power returns
shutdown.stop	Stop a shutdown in progress
shutdown.reboot	Shut down the load briefly while rebooting the UPS
shutdown.reboot.graceful	After a delay, shut down the load briefly while rebooting the UPS
test.panel.start	Start testing the UPS panel
test.panel.stop	Stop a UPS panel test
test.failure.start	Start a simulated power failure
test.failure.stop	Stop simulating a power failure
test.battery.start	Start a battery test
test.battery.start.quick	Start a "quick" battery test
test.battery.start.deep	Start a "deep" battery test
test.battery.stop	Stop the battery test
test.system.start	Start a system test
calibrate.start	Start runtime calibration
calibrate.stop	Stop runtime calibration
bypass.start	Put the UPS in Bypass mode
bypass.stop	Take the UPS out of Bypass mode
reset.input.minmax	Reset minimum and maximum input voltage status
reset.watchdog	Reset watchdog timer (forced reboot of load)
beeper.enable	Enable UPS beeper/buzzer
beeper.disable	Disable UPS beeper/buzzer
beeper.mute	Temporarily mute UPS beeper/buzzer
beeper.toggle	Toggle UPS beeper/buzzer
outlet.n.shutdown.return	Turn off the outlet possibly after a delay and return when power is back
outlet.n.load.off	Turn off the outlet immediately
outlet.n.load.on	Turn on the outlet immediately
outlet.n.load.cycle	Power cycle the outlet immediately
outlet.n.shutdown.return	Turn off the outlet and return when power is back

C.5.1 Experimental instant commands

The following commands were added to test feature support, but are not expected to last as part of NUT standard protocol in the long run.

Table 1: Vendor-dependent "ECO" modes

Name	Driver/Devices	Description
experimental.ecomode.enable	usbhid-ups ⇒ mge-hid (Eaton/MGE)	Put UPS in High Efficiency (aka ECO) mode
experimental.ecomode.disable	usbhid-ups ⇒ mge-hid (Eaton/MGE)	Take the UPS out of High Efficiency (aka ECO) mode
experimental.ecomode.start.auto	usbhid-ups ⇒ mge-hid (Eaton/MGE)	Put UPS in Bypass mode then High Efficiency (aka ECO) mode

Table 1: (continued)

Name	Driver/Devices	Description
experimental.ecomode.stop.auto	usbhid-ups ⇒ mge-hid (Eaton/MGE)	Take the UPS out of High Efficiency (aka ECO) mode after exiting Bypass mode
experimental.essmode.enable	usbhid-ups ⇒ mge-hid (Eaton/MGE)	Put UPS in Energy Saver System (aka ESS) mode
experimental.essmode.disable	usbhid-ups ⇒ mge-hid (Eaton/MGE)	Take the UPS out of Energy Saver System (aka ESS) mode

Currently the commands above are present in one subdriver and are specific to the vendor's proposed power state machine. The plan is to generalize the concept with vendor specifics, similarly to `experimental.ups.mode.buzzwords`.

D Hardware Compatibility List

Refer to the [online HCL](#).

E Documentation

E.1 User Documentation

- [FAQ - Frequently Asked Questions](#)
- [NUT user manual](#)
- [Cables information](#)
- [User manual pages](#)
- [Devices Dumps Library \(DDL\)](#): Provides information on how devices are supported; see also [the HCL](#)
- [Notes on NUT monitoring of USB devices in Solaris and related operating systems](#)
- [NUT Configuration Examples](#) book maintained by Roger Price
- [NUT GitHub Wiki](#)

E.2 Developer Documentation

- [NUT Developer Guide](#)
- [NUT Quality Assurance and Build Automation Guide](#)
- [NUT Packager Guide](#)
- [NUT Release Notes](#)
- [NUT Change Log](#)
- [UPS protocols library](#)
- [Developer manual pages](#)
- [NUT Quality Assurance](#)
- [Devices Dumps Library \(DDL\)](#): Provides simulation data to the `dummy-ups(8)` driver

E.3 Data dumps for the DDL

Note: both developers contributing a driver and users using an existing driver for device not previously documented as supported by it, are welcome to report new data for the Devices Dumps Library (DDL) mentioned above. Best of all such "data dump" reports can be prepared by the `./tools/nut-ddl-dump.sh` script from the main NUT codebase, and reported on the NUT mailing list or via [NUT issues on GitHub](#) or as a pull request against the [NUT Devices Dumps Library](#) following the naming and other rules described in the DDL documentation page.

Data dumps collected by the tools above, or by `upsc` client, or by drivers in exploratory data-dumping mode (with `-d 1` argument), can be compared by `./tools/nut-dumpdiff.sh` script from the main NUT codebase, which strips away lines with only numeric values (aiming to minimize the risk of losing meaningful changes like counters).

E.4 Offsite Links

These are general information about UPS, PDU, ATS, PSU and SCD:

- [UPS HOWTO](#) (The Linux Documentation Project)
- [UPS on Wikipedia](#)
- [PDU on Wikipedia](#)
- [Automatic Transfer Switch](#)
- [Power Supply Units](#)
- [Solar controller on Wikipedia](#)
- [UPS on The PC Guide](#)

These are writeups by users of the software.

- [NUT Configuration Examples and helper scripts](#) (*Roger Price*) (sources replicated in NUT GitHub organization as [ConfigExamples](#), [TLS-UPSmon](#), and [TLS-Shims](#))
 - [nut – testing the shutdown mechanism](#) (*Dan Langille*)
 - [Deploying NUT on an Ubuntu 10.04 cluster](#) (*Stefano Angelone*)
 - [Monitoring a UPS with nut on Debian or Ubuntu Linux](#) (*Avery Fay*)
 - [Installation et gestion d'un UPS USB en réseau sous linux](#) (*Olivier Van Hoof, french*)
 - [Network UPS Tools \(NUT\) on Mac OS X \(10.4.10\)](#) (*Andy Poush*)
 - [Interfacing a Contact-Closure UPS to Mac OS X and Linux](#) (*David Hough*)
 - [How to use UPS with nut on RedHat / Fedora Core](#) (*Kazutoshi Morioka*)
 - [FreeBSD installation procedure](#) (*Thierry Thomas, from FreeBSD*)
 - [Gestionando un SAI desde OpenBSD con NUT](#) (*Juan J. Martinez, spanish*)
 - [HOWTO: MGE Ellipse 300 on gentoo](#) (*nielchiano*)
 - [Cum se configurează un UPS Apollo seria 1000F pe Linux](#) (*deschis, Romanian*)
 - [Install a UPS \(nut\) on a Buffalo NAS](#) (*various authors*)
 - [NUT Korean GuideBook](#) (*PointBre*)
 - [USB UPS, notifications, and Home Assistant](#) (*James Ridgway*)
 - [PowerWalker UPS on Fedora Server 36](#) (*Michael Hirschler*)
-

Video articles are also available:

- [Network UPS Tools \(NUT Server\) Ultimate Guide](#) (*Techno Tim*)
- [NUT on my Pi, so my servers don't die](#) (*Jeff Geerling*) also including Home Assistant (HA) examples
 - [Jeff's lab tour due to an outage](#)
 - [Save your servers! NUT on a Raspberry Pi!](#)
 - [Ansible PiNUT project](#) and `geerlingguy.nut_client` Ansible role

E.5 News articles and Press releases

- [Linux UPS Without Tears](#) (*A. Lizard*)
- [Graceful UPS shutdowns on Linux](#) (*Carla Schroder*)

F Support instructions

There are various ways to obtain support for NUT.

F.1 Documentation

- First, be sure to read the [FAQ](#). The most common problems are already addressed there.
- Else, you can read the [NUT user manual](#). It also covers many areas about installing, configuring and using NUT. The specific steps on system integration are also discussed.
- Finally, [User manual pages](#) will also complete the User Manual provided information. At least, read the manual page related to your driver(s).

F.2 Mailing lists

If you have still not found a solution, you should search the lists before posting a question.

Someone may have already solved the problem:

[search on the NUT lists using Google](#)

Finally, you can **subscribe** to a NUT mailing list to:

F.2.1 Request help

Use the [NUT Users](#) mailing list.

In this case, be sure to include the following information:

- OS name and version,
- exact NUT version,
- NUT installation method: package, or a custom build from source tarball or GitHub (which fork, branch, PR),
- exact device name and related information (manufacturing date, web pointers, ...),
- complete problem description, with any relevant traces, like system log excerpts, and driver debug output. You can obtain the latter using the following command, running as root and after having stopped NUT:

```
/path/to/driver -DD -a <upsname>
```

If you don't include the above information in your help request, we will not be able to help you!

F.2.2 Post a patch, ask a development question, ...

Use the [NUT Developers](#) mailing list.

Refer to the [NUT Developer Guide](#) for more information, and the chapter on how to [submit patches](#).

Note that the currently preferable way for ultimate submission of improvements is to [post a pull request](#) from your GitHub fork of NUT. Benefits of PRs include automated testing and merge-conflict detection and resolution, as well as tracking discussion that is often needed to better understand, integrate or document the patch.

F.2.3 Discuss packaging and related topics

Use the [NUT Packagers](#) mailing list.

Refer to the [NUT Packager Guide](#) for more information.

F.3 IRC (Internet Relay Chat)

Yes, we're open!

There is an official `#nut` channel on <https://libera.chat/> network.

Feel free to hang out with whoever is on-line at the moment, or watch reports from the NUT CI farm as they come.

Please don't forget the basics of netiquette, such as that any help is done on a best-effort basis, people have other obligations, and are not always there even if their chat client is, and that respect and politeness are the norm (this includes doing some research before asking, and explaining the context where it is not trivial).

F.4 GitHub Issues

See <https://github.com/networkupstools/nut/issues> for another venue of asking (and answering) questions, as well as proposing improvements.

To report new Devices Dumps Library entries, posting an issue is okay, but posting a [pull request](#) is a lot better —easier for maintainers to review and merge any time. For some more detailed instructions about useful DDL reports, please see [NUT DDL page](#).

G Cables information

G.1 APC

G.1.1 940-0024C clone

From D. Stimits



Note

The original 940-0024C diagram was contributed by Steve Draper.

G.1.2 940-0024E clone

Reported by Jonathan Laventhol

This cable is said to use the same wiring as 940-0024C clone.

G.1.3 940-0024C clone for Macs

From Miguel Howard

**G.2 Belkin****G.2.1 OmniGuard F6C***-RKM**

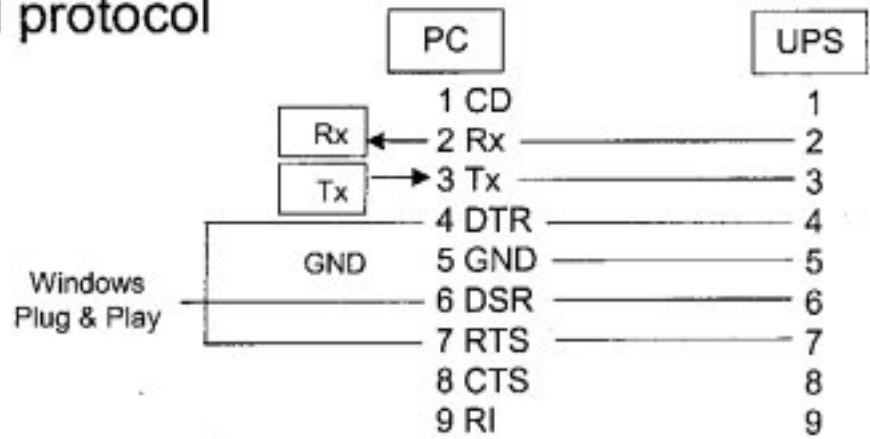
From "Daniel"

A straight-through RS-232 cable (with pins 2-7 connected through) should work with the following models:

- F6C110-RKM-2U
- F6C150-RKM-2U
- F6C230-RKM-2U
- F6C320-RKM-3U

● RS232

Character based protocol



G.3 Eaton

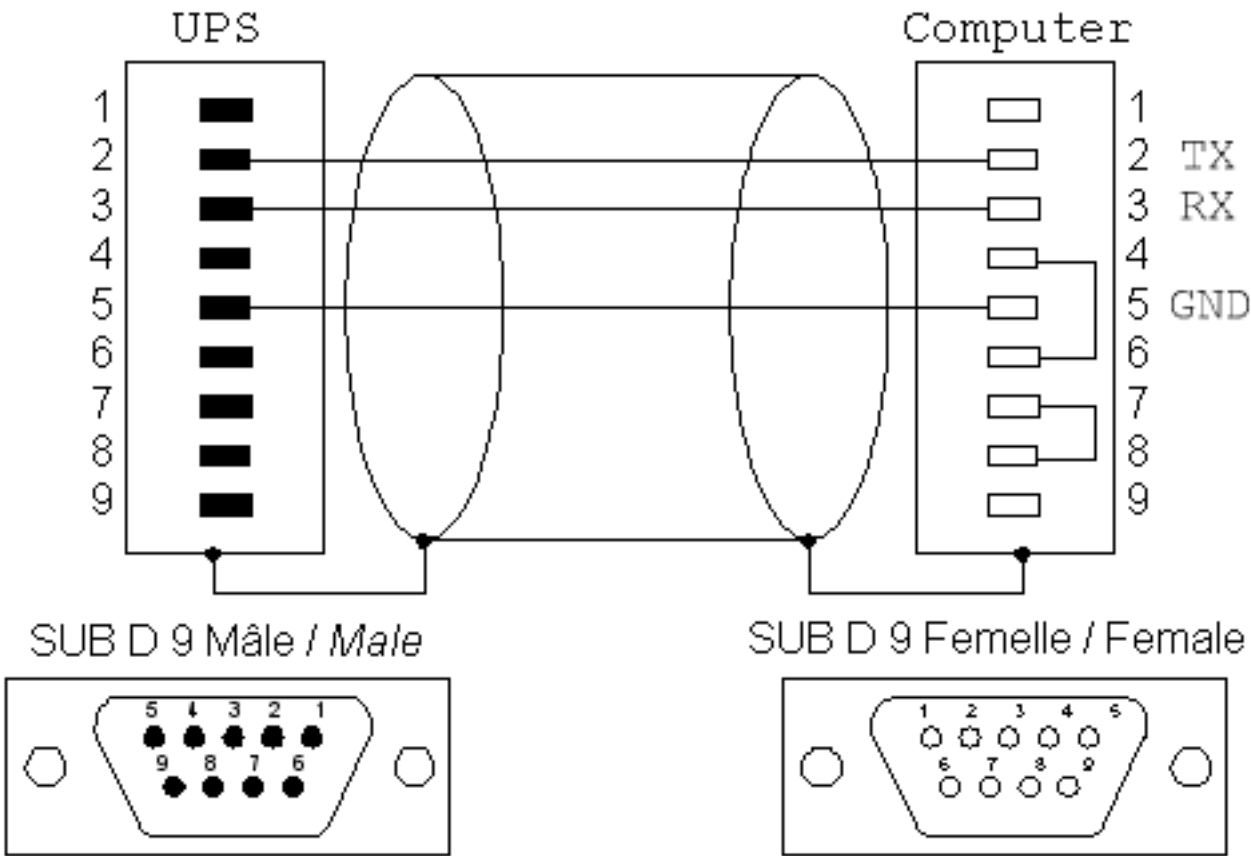
Documents in this section are provided courtesy of Eaton.

G.3.1 MGE Office Protection Systems

The three first cables also applies to MGE UPS SYSTEMS and Eaton.

DB9-DB9 cable (ref 66049)

This is the standard serial cable, used on most units.



DB9-RJ45 cable

This cable is used on the more recent models, including Ellipse MAX, Protection Station, ...



NMC DB9-RJ45 cable

The following applies to the MGE 66102 NMC (Network Management Card), and possibly other models. The NMC connection is an 8P8C RJ45-style jack.

Signal	PC	NMC
	1,4,6	
TxD	2	3
RxD	3	6
GND	5	4
	7,8	
	shield	shield

USB-RJ45 cable

This cable is used also on the more recent models, including Ellipse MAX, Protection Station, ...

CABLE :



DB9-RJ12 cable

This cable is used on some older Ellipse models.



G.3.2 Powerware LanSafe

LanSafe for
3115 / 5105 / 5119 / 9110 / 9150 / 9305



G.3.3 SOLA-330

Just uses a normal serial cable, with pin 1-1 through to 9-9.



G.4 HP - Compaq

G.4.1 Older Compaq UPS Family

This cable can be used with the following models:

T700, T1000, T1500, T1500j, T700h, T1000h, T1500h, R1500, R1500j, R1500h, T2000, T2000j, T2400h, T2400h-NA, R3000 / R3000j, R3000h, R3000h-International, R3000h-NA, R6000h-NA, R6000i, R6000j.

UPS	PC	9 pin connector
1 -----	3	
2 -----	2	
	4 -\	
4 -----	5	
	6 -/	
6 -----	7	

Contributed by Kjell Claesson and Arnaud Quette.

G.5 Phoenixtec (Best Power)

Many Best Power units (including the Patriot Pro II) have a female DB-9 socket with a non-standard pinout.

Signal	PC	UPS
	1,4,6	NC
TxD	2	2
RxD	3	1
GND	5	4
	7,8	NC

Sources:

- http://pinoutsguide.com/UPS/best_power_pinout.shtml

- http://lit.powerware.com/lit_download.asp?file=m_patriotproii_jan99.pdf
- Stan Gammons

G.6 Tripp-Lite

From Tripp-Lite, via Bryan Kolodziej

This cable (black 73-0844 cable) is used on various models, using the "Lan 2.2 interface" and the genericups driver (upstype=5).



H Configure options

Note

For more information about build environment setup, see chapters about [Prerequisites for building NUT on different OSes](#) (or docs/config-prereqs.txt in NUT sources for up-to-date information) and [Custom NUT CI farm build agents: LXC multi-arch containers](#) (or docs/ci-farm-lxc-setup.txt in NUT sources for up-to-date information).

As many other projects relying on GNU autotools for build recipe automation, NUT sources deliver a `configure` script which is used to set up numerous nuances relevant for your build of the project. Most of the configuration requirements are passed by `--with-something` or `--enable-something` options (allegedly, not always used consistently with autotools naming recommendations), and with environment variables like `CFLAGS` typically also passed as command line options.

Default syntax of `--with-something` or `--enable-something` assumes a boolean approach, so the option as such conveys a `yes` string value, and aliases `--without-something` or `--disable-something` are handled automatically as a `no` string value, for the option named `something`.

In many cases, these options are used to pass non-boolean configuration of the option in question, commonly a path, program name, compiler flags to use along with a particular dependency, user name or type of documentation to build, etc.

A special case here concerns options that accept an `auto` value, where the `configure` script would opine to request a final `yes` or `no`, depending on other circumstances of the build environment and requested configuration.

Note

When building NUT from Git sources rather than the distribution tarballs, you would have to generate the `configure` script and some further needed files by running `./autogen.sh`. This in turn may require additional tools and other dependencies on your build environment (referenced just a bit below). For common developer iterations, porting to new platforms, or in-place testing, running the `./ci_build.sh` script can be a helpful one-stop solution.

The NUT [Packager Guide](#), which presents the best practices for installing and integrating NUT, is also a good reading.

The [Prerequisites for building NUT on different OSes](#) (or `docs/config-prereqs.txt` in NUT sources for up-to-date information) document suggests prerequisite packages with tools and dependencies available and needed to build and test as much as possible of NUT on numerous platforms, written from perspective of CI testing (if you are interested in getting updated drivers for a particular device, you might select a sub-set of those suggestions).

There are a few options reviewed below that can be given to `configure` script to tweak your compilations. See also `./configure --help` for a current and complete listing for the current version of the codebase.

NUT tracks `configure` options used during build, so you can view them to produce a replacement by asking NUT programs for `--help` or for `--version` with debugging enabled (first), e.g.:

```
;; upsd -DV
Network UPS Tools upsd 2.8.1
Network UPS Tools version 2.8.1 configured with flags: --with-all=auto --with-doc=skip ...
```

A more industrial approach is to use `lib/libupsclient-config --config-flags`, where supported. Note that the `pkg-config` manifest `libupsclient.pc` does not easily convey this information.

H.1 Keeping a report of NUT configuration

`--enable-keep_nut_report_feature`

If this option is enabled (not currently default), the report displayed by `configure` script will be kept in a file when the script ends, and installed into the data directory.

H.2 In-place replacement defaults

A common situation for NUT builds is to verify whether current version of the codebase (e.g. recent and not-yet-packaged release, or a Git branch) solves some issues of an existing deployment. Such tests are simplified if the new build of NUT plays by the same systems integration rules as the already deployed (e.g. package-delivered) version, specifically about filesystem access permissions and configuration file locations.

`--enable-inplace-runtime`

Tries to detect and pre-set `configure` defaults for run-time settings (which you can still override if needed, but no longer **must** specify explicitly to be on same page as the existing setup), most notably:

- `--sysconfdir`
 - `--with-user`
-

- `--with-group`

If the installed NUT version supports reporting of `CONFIG_FLAGS` used during its build, the `configure` script will try to take those values into account when running in this mode (and so use the same program and data installation paths, for example).

Note

This does not currently rely on the configuration report optionally installed by `--enable-keep_nut_report_feature` above, but might do so eventually.

H.3 Driver selection

H.3.1 Serial drivers

`--with-serial`

H.3.2 USB drivers

Build and install the serial drivers (default: yes)

`--with-usb`

Build and install the USB drivers (default: auto-detect)

Note that you need to install the `libusb` development package or files, and that both `libusb 0.1` and `1.0` are supported. In case both are available, `libusb 1.0` takes precedence, and will be used by default.

It is however possible to override this default choice by explicitly calling `--with-usb=libusb-0.1` or `--with-usb=libusb-1.0`.

If you do specify the version to use (or `yes` for auto-detection), this option would fail if requested (or any) `libusb` version was not found. The default `auto` value would not fail in such case.

**Warning**

If you intend to use the `libmodbus` variant with `libusb` support, you would require `libusb-1.0` specifically; the implementation or "compat API" of `0.1` is not supported by that library version.

H.3.3 SNMP drivers

`--with-snmp`

Build and install the SNMP drivers (default: auto-detect)

Note that you need to install `libsnmp` development package or files.

`--with-net-snmp-config`

In addition to the `--with-snmp` option above, this one allows to provide a custom program name (in `PATH`) or complete path-name to `net-snmp-config` (may have copies named per architecture, e.g. `net-snmp-config-32` and `net-snmp-config-64`).

This may be needed on build systems which support multiple architectures, or in cases where your distribution names this program differently. With a default value of `yes` it would mean preference of this program, compared to information from `pkg-config`, if both are available.

H.3.4 XML drivers and features

`--with-neon`

Build and install the XML drivers (default: auto-detect)

Note that you need to install neon development package or files.

H.3.5 LLNC CHAOS Powerman driver

`--with-powerman`

Build and install Powerman PDU client driver (default: auto-detect)

This allows to interact with the Powerman daemon, and the numerous Power Distribution Units (PDU) supported by the [powerman](#) project.

Note that you need to install powerman development package or files.

H.3.6 IPMI drivers

`--with-ipmi`
`--with-freeipmi`

Build and install IPMI PSU driver (default: auto-detect)

This allows to monitor numerous Power Supply Units (PSU) found on servers.

Note that you need to install freeipmi (0.8.5 or higher, for nut-scanner; and 1.0.1 or higher, for nut-ipmipsu) development package or files.

H.3.7 I2C bus drivers

`--with-linux_i2c`

Build and install i2c drivers (default: auto-detect)

Note that you need to install libi2c development package or files.

H.3.8 GPIO bus drivers

`--with-gpio`

Build and install GPIO drivers (default: auto-detect)

Note that on Linux you need to install libgpiod library and development package or files. This seems to be present in distributions released after roughly 2018. Other platforms are not currently supported, but may be in the future.

H.3.9 Modbus drivers

`--with-modbus`

Build and install modbus (Serial, TCP) drivers (default: auto-detect)

Note that you need to install libmodbus development package or files.

```
--with-modbus+usb
```

Require a variant of libmodbus with RTU USB support. This feature is currently not available in upstream project or OS distribution packages, so your NUT build environment should provide a prerequisite build of https://github.com/networkupstools/libmodbus/tree/rtu_usb (may be a static library build, used from a temporary installation prefix location, to avoid potential conflicts with the OS packaged shared library). You would also need specifically `libusb-1.0` (not the older API).

At the time of this writing, such constraint can be desirable for the `apc_modbus(8)` driver which supports different communication media.

For more details please see <https://github.com/networkupstools/nut/wiki/APC-UPS-with-Modbus-protocol>

H.3.10 Manual selection of drivers

```
--with-drivers=<driver>,<driver>,...
```

Specify exactly which driver or drivers to build and install (this works for serial, usb, and snmp drivers, and overrides the preceding three options).

As of the time of original writing (2010), there are 46 UPS drivers available. Most users will only need one, a few will need two or three, and very few people will need all of them.

To save time during the compile and disk space later on, you can use this option to just build and install a subset of the drivers. For example, to select `mge-shut` and `usbhid-ups`, you'd do this:

```
--with-drivers=apcsmart,usbhid-ups
```

If you need to build more drivers later on, you will need to rerun `configure` with a different list. To make it build all of the drivers from scratch again, run `make clean` before starting.

H.4 Optional features

H.4.1 CGI client interface

```
--with-cgi (default: no)
```

Build and install the optional CGI programs, HTML files, and sample CGI configuration files. This is not enabled by default, as they are only useful on web servers. See <data/html/README> for additional information on how to set up CGI programs.

H.4.2 NUT Scanner tool

```
--with-nut-scanner (default: auto)
```

Build and install the optional `nut-scanner(8)` tool to discover some types of UPS or ePDU devices on locally or remotely connected media (such as Serial, USB, SNMP, IPMI, NetXML), as well as NUT data servers (by Avahi/mDNS broadcasts or "old" NUT port polling) and simulation device configurations that can be relayed by a local `dummy-ups(8)` driver instance.

Many of the features depend on third-party libraries that are loosely linked at run-time using `libltdl`, and due to that, the build, delivery and use of the tool does not depend on **all** of them being available in the final deployment.

H.4.3 NUT Configuration tool

```
--with-nutconf (default: auto)
```

Build and install the optional `nutconf(8)` tool to create and manipulate NUT configuration files. It also optionally supports device scanning, using the `nutscan(3)` library (if built), to suggest configuration of devices.

H.4.4 Pretty documentation and man pages

```
--with-docs=<output-format(s)>  (default: no)
--with-doc=<output-format(s)>    (default: no)
```

Build and install NUT documentation file(s). Note: `--with-doc` is a legacy NUT option, and `--with-docs` is an alias that matches the more wide-spread equivalent in different packaging frameworks and projects.

This feature requires AsciiDoc 8.6.3 or newer (see <https://asciidoc.org>).

The possible documentation type values are:

- `html-single` for single page HTML,
- `html-chunked` for multi-paged HTML,
- `pdf` for a PDF file, and
- `man` for the usual man pages.

Other values understood for this option are listed below:

- If the `--with-doc` argument is passed without a list, or specifies just `=yes` or `=all`, it enables all supported formats with a `=yes` to require them.
- An (explicit!) `--with-doc=auto` argument tries to enable all supported formats with an `=auto` but should not fail the build if something can not be generated.
- A `--with-doc=no` quietly skips generation of all types of documentation, including man pages.
- A `--with-doc=skip` is used to configure some of the `make distcheck*` scenarios to re-use man page files built and distributed by the main build and not waste time on re-generation of those.
- A `--with-doc=dist-auto` allows to use pre-distributed MAN pages if present (should be in "tarball" release archives; should not be among Git-tracked sources; may be left over from earlier builds in same workspace), or build those if we can (the `auto` part). Practically this is implemented in detail only for `--with-doc=man=dist-auto`, as we do not dist HTML and PDF products; it is a placeholder for those to simplify the generic configuration calls.

Multiple documentation format values can be specified, separated with comma. Each such value can be suffixed with `=yes` to require building of this one documentation format (abort configuration if tools are missing), `=auto` to detect and enable if we can build it on this system (and not abort if we can not), and `=no` (or `=skip`) to explicitly skip generation of this document format even if we do have the tools to build it.

If a document format is mentioned in the list without a suffix, then it is treated as a `=yes` requirement.

Verbose output can be enabled using: `ASCIIDOC_VERBOSE=-v make`

Example valid formats of this flag:

- `--with-doc` without an argument, effectively same as `--with-doc=yes`
- `--with-doc=` is a valid empty list, effectively same as `--with-doc=no`
- `--with-doc=auto`
- `--with-doc=pdf,html-chunked`
- `--with-doc=man=no,pdf=auto,html-single`

Additional flags with regard to documentation generation include:

```

--with-docs-man-section-api=<SN>          (default: 3)
--with-docs-man-section-cfg=<SN>          (default: 5)
--with-docs-man-section-cmd-sys=<SN>      (default: 8)
--with-docs-man-section-cmd-usr=<SN>      (default: 1)

```

Customize man page section identifiers for operating systems whose standards do not match defaults listed above for Library and API, Configuration Files, System Management Commands and User Commands sections (e.g. Solaris-derived systems might use `--with-docs-man-section-cmd-sys=1m`). This impacts not only file names of the manual pages, but also cross-links in generated documents. Note that use of these flags does not implicitly enable any `--with-docs=...` mode, which should still be specified explicitly.

H.4.5 Python support

NUT includes a client binding `PyNUT` module, as well as an optional GUI application `NUT-Monitor` (not to be confused with `nut-monitor` service name for `upsmon` as delivered by some OS distribution packages). Both python 2.7 and several versions of python 3.x are supported and regularly tested by NUT CI farm builds; other versions may work or not.

Also some of the source configuration (including activity in `autogen.sh` script and later in certain `Makefile`s` during build) relies on presence of ``python` (and `perl`) interpreters. If they are not available, e.g. on older operating systems, certain features are skipped—but you may have to export special environment variables to signal to `autogen.sh` that this is an expected situation (it would suggest which, for your current NUT version).

Note

For CI builds (or similar reproducible developer activity) performed with `ci_build.sh`—since this is an area which impacts both `configure` script generation by the helper `autogen.sh` script and subsequently the choices made by the `configure` script itself, settings can be made by exporting the `PYTHON` environment variable which should evaluate to some way to call the correct interpreter (e.g. `python2.7`, `/usr/bin/env python` or `/usr/bin/python3`).

The `configure` script does support equivalent options, whose defaults come from detection of certain program names by current `PATH` setting. Further use of these interpreter names and other paths during NUT build and installation is managed by `Makefile` variable expansion, as prepared by the `configure` script.

Also note that it is not required to use the same `PYTHON` implementation for `autogen.sh` to do its job, and for the `configure` script option.

As noted above, both `python-2.x` and `python-3.x` variants are supported. You can consult the `m4/nut_check_python.m4` file for detection methods used. If both interpreter generations are present, and a particular un-versioned `PYTHON` is not specified or detected, then selected/detected `PYTHON3` is chosen as the ultimate `PYTHON` value.

For majority of uses in the build procedure and products, the generation of Python does not matter and the un-versioned `PYTHON` value is substituted into files as the script shebang, used to find the `site-packages` location, etc.

One exception is generation of NUT-Monitor GUI application which has been separated for `NUT-Monitor-py2gtk2` and `NUT-Monitor-py3qt5` due to further backend platform technical differences—these build products specifically use `PYTHON2` and `PYTHON3` substitutions. They may be co-installed on the same system. A dispatcher shell script `NUT-Monitor` is used to launch the preferred (newest) or the only existing implementation.

Please note that by default NUT tries to make use of everything in your build environment, so if both Python generation are detected—the binding module will be delivered into both, and two versions of NUT-Monitor GUI application will be installed. If you want to avoid that behaviour on a build system with both interpreters present, you can explicitly specify to build e.g. `--without-python2 --with-python=/usr/bin/python-3.9`.

The settings below may be of particular interest to non-distribution packaging efforts with their own dedicated directory trees:

```
--with-python=SHEBANG_PATH
```

Specify a definitive version you want used for majority of the Python code (except version-dependent scripts, see above).

The SHEBANG_PATH should be a full program pathname, optionally with one argument, e.g. `/usr/bin/python-3.9` or `/usr/bin/env python2`.

Defaults (in order): * `python`, `python3` or `python2` program if present in `PATH` by such name, * or the newest of `PYTHON3` or `PYTHON2` values (specified or detected below).

```
--with-python2=SHEBANG_PATH
--with-python3=SHEBANG_PATH
```

For version-dependent scripts (see above) or to default the newest Python version if not specified by `--with-python` option or detected otherwise, you can provide the preferred version and implementation of Python 2 or 3 respectively.

Conversely, if neither of these configure options were specified, but some `--with-python` program was specified or detected, and its report says it has Python major version 2 or 3, then the versioned interpreter string would point to that.

```
--with-nut_monitor
```

Install the NUT-Monitor GUI application (depending on Python 2 or 3 version availability), and optional `desktop-file-install` integration).

```
--with-pynut
```

Install the PyNUT module files for general consumption into "site-packages" location of the currently chosen Python interpreter(s): yes, no, auto. or dedicated as the required dependency of NUT-Monitor application (app).

Warning



The module files are installed into a particular Python version's location such as `/usr/lib/python2.7/dist-packages` even if you specify a relaxed or un-versioned interpreter like `python2` (which would be used in scripts for the NUT-Monitor application and possible other consumers of the module). If the preferred Python version in the deployed system changes later (so the `python2` symlink for this example would point elsewhere) the module import would become not resolvable for such consumers, until it is installed into that other Python's "site-packages" location.

H.4.6 Development files

```
--with-dev (default: no)
```

Build and install the `upsclient` and `nutclient` library and header files, to build further projects against NUT (such as `wmNUT` client and many others).

Also delivers `libnutscan` library and header files, if `--with-nut-scanner` is enabled.

Disabled development file delivery, in particular, disables installation of static libraries.

```
--with-dev-libnutconf (yes/no/auto, default: no)
```

Build and install the `nutconf` library and header files, to build further projects against NUT (especially those that need to set it up). This is currently not automatically tied into either `--with-dev` nor `--with-nutconf` (note below on `--with-all`), because the tool and library are experimental and the API may yet undergo significant changes in the coming years.

It is more recommended to use the CLI tool as a presumably more stable API for third-party integrations, than the library directly.

This feature may however be enabled or disabled implicitly, using `--with-all` (=yes/no/auto) request, based on a combination of `--with-nutconf` and `--with-dev` options:

- if `--without-all`, default to set `--with-dev-libnutconf=no`; otherwise...
-

- if either `--without-dev` or `--without-nutconf`, default to set `--with-dev-libnutconf=no`; otherwise...
- if either `--without-dev=auto` and/or `--without-nutconf=auto`, default to set `--with-dev-libnutconf=auto`; otherwise...
- if both `--with-dev` and `--with-nutconf`, default to set `--with-dev-libnutconf=yes`.

Requires C++11 support to be enabled (available in compiler, not neutered by explicit selection of an older standard).

```
--enable-ldflags-nut-rpath (yes/no/auto/flags, default: auto)
--enable-ldflags-nut-rpath-cxx (yes/no/auto/flags, default: auto)
```

NUT development files include the `*.pc` metadata for `pkg-config` ecosystem and a legacy `libupsclient-config` script, both used to deliver compiler and linker options for third-party programs to build against NUT libraries.

There can be a problem during run-time, when NUT is installed into a location that the system linker does not know to look into, e.g. the default `--prefix` value of `/usr/local/ups`, that the built programs fail to dynamically link with the shared objects of NUT libraries, unless deprecated tricks like the `LD_LIBRARY_PATH` setting are applied.

These options allow NUT to add linker options, which set "RPATH" in consumer binaries, to the announced metadata. As a result, third-party programs or libraries are hard-coded to look for NUT libraries in a certain location, in addition to system locations and similar paths added by other dependencies.

By default (in `auto` mode), the `configure` script queries the compiler for known system library paths (currently supported with GCC and CLANG builds), and checks the NUT `libdir` against that list. If there is a hit, no extra flags are announced; otherwise, the format detected for currently used linker by `autoconf` `libtool` macros is added to the announced metadata for builds against NUT. If this causes problems, you can either `--disable-...` these flags or provide the desired string explicitly.

H.4.7 Options for developers

```
--enable-spellcheck (default: auto)
```

Activate recipes for documentation source file spelling checks with `aspell` tool. Default behavior depends on availability of the tool, so if present — it would run for `make check` by default, to facilitate quicker acceptance of contributions.

```
--enable-check-NIT (default: no)
```

Add `make check-NIT` to default activity of `make check` to run the NUT Integration Testing suite. This is potentially dangerous (e.g. due to port conflicts when running many such tests in same environment), so not active by default.

```
--enable-maintainer-mode (default: no)
```

Use `maintainer` mode to keep `Makefile.in` and `Makefile` in sync with ever-changing `Makefile.am` content after Git updates or editing.

```
--enable-cppcheck (default: no)
```

Activate recipes for static analysis with `cppcheck` tools (if available).

```
--with-unmapped-data-points (default: no)
```

Build SNMP and USB-HID subdrivers with entries discovered by the scripts which generated them from data walks, but developers did not rename yet to NUT mappings conforming to `docs/nut-names.txt` standards. Production driver builds must not include any non-standard names.

```
--enable-NUT_STRARG-always (default: auto)
```

Enable the `NUT_STRARG` macro (to handle NULL string printing) even if the system libraries seem to safely support this behavior natively? This flag should primarily help against overly zealous static analysis tools in recent compiler generations. The `auto` value enables the flag for certain compiler versions known to complain about some of our use cases — even though those seem to be false positives.


```
--enable-configure-debug
```

Cause the `configure` script run to report some internal values of variables as it goes through making of choices, to help recipe developers troubleshoot it.

H.4.8 I want it all!

```
--with-all (no default)
```

Build and install all of the above (the serial, USB, SNMP, XML/HTTP and PowerMan drivers, the CGI programs and HTML files, and the upsclient library).

You can also use `--with-all=auto` to detect available prerequisites and only build everything that we can build on this system (but not fail due to inability to request a build of something from the full scope).

H.4.9 Networking transport security

```
--with-ssl (default: auto-detect)
--with-nss (default: auto-detect)
--with-openssl (default: auto-detect)
```

Enable SSL support, using either Mozilla NSS or OpenSSL.

If both are present, and nothing was specified, OpenSSL support will be preferred.

Read [docs/security.txt](#) for instructions on SSL support.

Note

Currently the two implementations differ in supported features.

H.4.10 Networking access security

```
--with-wrap (default: auto-detect)
```

Enable libwrap (tcp-wrappers) support.

Refer to [upsd\(8\)](#) man page for more information.

H.4.11 Networking IPv6

```
--with-ipv6 (default: auto-detect)
```

Enable IPv6 support.

H.4.12 AVAHI/mDNS

```
--with-avahi (default: auto-detect)
```

Build and install Avahi support, to publish NUT server availability using mDNS protocol. This requires Avahi development files for the Core and Client parts.

H.4.13 LibLTDL

```
--with-libltdl (default: auto-detect)
```

Enable libltdl (Libtool dlopen abstraction) support.

This is required to build `nut-scanner` which loads third-party libraries dynamically, based on requested scanning options. This allows to build and package the tool without requiring all possible dependencies to be installed in each run-time environment.

H.5 Other configuration options

H.5.1 NUT data server port

```
--with-port=PORT
```

Change the TCP port used by the network code. Default is 3493 as registered with IANA.

Ancient versions of `upsd` used port 3305. NUT 2.0 and up use a substantially different network protocol and are not able to communicate with anything older than the 1.4 series.

If you have to monitor a mixed environment, use the last 1.4 version, as it contains compatibility code for both the old "REQ" and the new "GET" versions of the protocol.

H.5.2 Daemon user accounts

```
--with-user=<username>
--with-group=<groupname>
```

See also `--enable-inplace-runtime`.

Programs started as `root` will `setuid()` to `<username>` for somewhat safer operation. You can override this with `-u <otheruser>` in several programs, including `upsd` (and all drivers by extension), `upsd`, and `upsmon`. The "user" directive in `ups.conf` overrides this at run time for the drivers.

Note

`upsmon` does not totally drop `root` because it may need to initiate a shutdown. There is always at least a stub process remaining with `root` powers. The network code runs in another (separate) process as the new user.

The `<groupname>` is used for the permissions of some files, particularly the hotplugging rules for USB. The idea is that the device files for any UPS devices should be readable and writable by members of that group.

The default value for both the username and groupname is `nobody` (or `nogroup` on systems that have it when `configure` script runs). This was done since it's slightly better than staying around as `root`. Running things as `nobody` is not a good idea, since it's a hack for NFS access. You should create at least one separate user for this software.

If you use one of the `--with-user` and `--with-group` options, then you have to use the other one too.

See the [INSTALL.nut](#) document and the FAQ for more on this topic.

H.5.3 Syslog facility

```
--with-logfacility=FACILITY
```

Change the facility used when writing to the log file. Read the man page for `openlog` to get some idea of what's available on your system. Default is `LOG_DAEMON`.

H.5.4 External API integration scripts

Sometimes as a developer or user you need to interact with a device for which a "proper" NUT driver does not yet exist (or is not in your version), but some proof-of-concept script can be good enough to collect some data.

In some cases, an UPS does not support local monitoring at all, but has a network port for cloud-based monitoring through its vendor's portal.

Such data can be converted and fed into the NUT `dummy-ups` driver, and so represented in the NUT ecosystem, by rewriting the "sequence" file whose contents it processes in a loop (see [dummy-ups\(8\)](#) for more details).

NUT provides sample scripts for such integration, which can be used if you have a suitable use-case, or provide inspiration for you to begin experiments with a new device and (as often happens) a shell or Python script polling it for information. For more details, see `scripts/external_apis` in NUT sources (and pull requests with more integrations would be welcome there).

```
--enable-extapi-enphase=(yes|auto|no)
```

Enable installation of integration script for External API: Enphase Monitor (default: no)

H.6 Installation directories

```
--prefix=PATH
```

This is a fairly standard option with GNU autoconf, and it sets the base path for most of the other install directories. The default is `/usr/local/ups`, which puts everything but the state sockets in one easy place, and does not conflict with usual distribution packaging.

If you like having things to be at more of a "system" level, setting the prefix to `/usr/local` or even `/usr` might be better.

```
--exec_prefix=PATH
```

This sets the base path for architecture dependent files. By default, it is the same as `<prefix>`.

```
--sysconfdir=PATH
```

Changes the location where NUT's configuration files are stored. By default this path is `<prefix>/etc`. Setting this to `/etc/nut` or `/etc/ups` might be useful. See also `--enable-inplace-runtime`.

The `NUT_CONFPATH` environment variable overrides this at run time.

```
--sbindir=PATH
```

```
--bindir=PATH
```

Where executable files will be installed. Files that are normally executed by root (`upsd`, `upsmon`, `upssched`) go to `<sbindir>`, all others to `<bindir>`. The defaults are `<exec_prefix>/sbin` and `<exec_prefix>/bin` respectively.

See also `--with-drivpath` below.

```
--with-drivpath=PATH
```

The UPS drivers will be installed to this path. By default they install to `<exec_prefix>/bin`, i.e. `/usr/local/ups/bin`.

You would want a location that remains mounted when most of the system is prepared to turn off, so some distributions package NUT drivers into `/lib/nut` or similar. See [config-notes.txt](#) detailing how to set up system shutdown.

The `driverpath` global directive in the `ups.conf` file overrides this at run time.

```
--datadir=PATH
```

Change the data directory, i.e., where architecture independent read-only data for the currently built project is installed. By GNU Autotools default, this is `<prefix>/share` (same as the system-wide "data root" which we also consult for third-party data such as Augeas lens delivery locations), i.e. `/usr/local/ups/share`.

+ Typically this is reconfigured to a `'${datarootdir}/nut'` value.

+ At the moment, this directory only holds two files by default — the optional `cmdvartab` and `driver.list`, but may hold additional data on certain systems (e.g. FreeBSD quirks, Solaris SMF or init methods, sometimes documentation).

`--mandir=PATH`

Sets the base directories for the man pages. The default is `<prefix>/man`, i.e. `/usr/local/ups/man`.

`--includedir=PATH`

Sets the path for include files to be installed when `--with-dev` is selected. For example, `upsclient.h` is installed here. The default is `<prefix>/include`.

`--libdir=PATH`

Sets the installation path for libraries. Depending on the build configuration, this can include the `libupsclient`, `libnutclient`, `libnutclientsub`, `libnutscan` and their `pkg-config` metadata (see `--with-pkgconfig-dir` option). The default is `<exec_prefix>/lib`.

`--libexecdir=PATH`

Sets the installation path for "executable libraries" — helper scripts or programs that are not intended for direct and regular use by people, and rather are implementation details of services. Depending on the build configuration, this can include the `nut-driver-enumerator.sh`, `sockdebug`, and others. The default is `<exec_prefix>/libexec`.

Package distributions may want to use this option to customize this path to include the package name, e.g. set it to `<exec_prefix>/libexec`.

`--with-pkgconfig-dir=PATH`

Where to install `pkg-config *.pc` files. This option only has an effect if `--with-dev` is selected, and causes a `pkg-config` file to be installed in the named location. The default is `<exec_prefix>/pkgconfig`.

Use `--without-pkgconfig-dir` to disable this feature altogether.

`--with-cgipath=PATH`

The CGI programs will be installed to this path. By default, they install to `<exec_prefix>/cgi-bin`, which is usually `/usr/local/ups/cgi-bin`.

Note

If you set the prefix to something like `/usr`, you should set the `cgipath` to something else, because `/usr/cgi-bin` is pretty ugly and non-standard.

The CGI programs are not built or installed by default. Use `./configure --with-cgi` to request that they are built and installed.

`--with-htmlpath=PATH`

HTML files will be installed to this path. By default, this is `<prefix>/html`. Note that HTML files are only installed if `--with-cgi` is selected.

`--with-hotplug-dir=PATH`

Where to install Linux 2.4 hotplugging rules. The default is to use `/etc/hotplug`, if that directory exists, and to not install it otherwise. Note that this installation directory is not a subdirectory of `<prefix>` by default. When installing NUT as a non-root user, you may have to override this option.

Use `--without-hotplug-dir` to disable this feature altogether.

`--with-udev-dir=PATH`

Where to install Linux 2.6 hotplugging rules, for kernels that have the "udev" mechanism. The default is to use `/etc/udev`, if that directory exists, and to not install it otherwise. Note that this installation directory is not a subdirectory of `<prefix>` by default. When installing NUT as a non-root user, you may have to override this option.

Use `--without-udev-dir` to disable this feature altogether.

`--with-systemdsystemunitdir=PATH`

Where to install Linux systemd unit definitions. Useless and harmless on other OSes, including Linux distributions without systemd, just adding a little noise to configure script output.

Use `--with-systemdsystemunitdir=auto` (default) to detect the settings using pkg-config if possible.

Use `--with-systemdsystemunitdir(=yes)` to require detection of these settings with pkg-config, or fail configuration if not possible.

Use `--with-systemdsystemunitdir=no` to disable this feature altogether.

`--with-systemdsystempresetdir=PATH`

Where to install Linux systemd unit presets (lists of services enabled or disabled by default). Useless and harmless on other OSes, including Linux distributions without systemd, just adding a little noise to the configure script output.

Use `--with-systemdsystempresetdir=auto` (default) to detect the settings using pkg-config if possible.

Use `--with-systemdsystempresetdir(=yes)` to require detection of these settings with pkg-config, or fail configuration if not possible.

Use `--with-systemdsystempresetdir=no` to disable this feature altogether.

`--with-systemdshutdownmdir=PATH`

Where to install Linux systemd unit definitions for shutdown handling. Useless and harmless on other OSes, including Linux distributions without systemd, just adding a little noise to configure script output.

Use `--with-systemdshutdownmdir` to detect the settings using pkg-config.

Use `--with-systemdshutdownmdir=no` to disable this feature altogether.

`--with-systemdtmpfilesdir=PATH`

Where to install Linux systemd configuration for tmpfiles handling (the automatically created locations for PID, state and similar run-time files). Useless and harmless on other OSes, including Linux distributions without systemd, just adding a little noise to configure script output.

Use `--with-systemdtmpfilesdir` to detect the settings using pkg-config.

Use `--with-systemdtmpfilesdir=no` to disable this feature altogether.

`--with-libsystemd=(auto|yes|no)`

`--with-libsystemd-includes=CFLAGS`

`--with-libsystemd-libs=LDFLAGS`

If the build system provides `libsystemd` headers, NUT binaries can be built with tighter integration to this service management framework. In this case NUT daemons (`upsd`, `upsmon`, `upslog` and `drivers`) would report their life-cycle milestones (READY, RELOADING, STOPPING) and support the watchdog reports (if enabled in their respective units by end-user — not done by default since the numbers depends on monitoring system performance). Default: "auto" (integration enabled if detected).

`--with-augeas-lenses-dir=PATH`

Where to install Augeas configuration-management lenses.

Only useful and valid if you use Augeas to parse and modify configuration files. The default is to use `/usr/share/augeas/lenses` if that directory exists, and to not install it otherwise.

H.7 Directories used by NUT at run-time

`--with-pidpath=PATH`

Changes the directory where NUT pid files are stored for processes running as `root`. By default this is `/var/run`. Certain programs like `upsmon` will leave files here.

`--with-altpidpath=PATH`

Programs that normally don't have `root` powers, like the drivers and `upsd`, write their PID files here. By default this is whatever the `statepath` (below) is, as those programs should be able to write there.

The `NUT_ALTPIDPATH` environment variable overrides this at run time.

`--with-statepath=PATH`

Change the default location of the local Unix sockets created by the drivers to interact with the data server `upsd` to report their state and receive commands. Default is `/var/state/ups`.

This is also the default location for non-`root` daemons to write a PID file, if a separate location is not specified by `--with-altpidpath` option.

The `NUT_STATEPATH` environment variable overrides this at run time.

Note

Fun fact: in early iterations of the NUT project, the drivers and the data server did exchange information by writing and reading complete state files in a commonly accessible location, hence the name.

`--with-powerdownflag=FILEPATH`

Change the default location (full filename path) of the `POWERDOWNFLAG` created by `upsmon` (as `root`) to tell the late-shutdown integration that this machine should tell all UPSes for which it is a "primary" NUT server to cut power to the load. Default is `/etc/killpower` on POSIX systems and `"C:\\killpower"` (note the double backslashes) on Windows.

H.8 Things the compiler might need to find

H.8.1 pkg-config tooling

`--with-pkg-config`

This option allows to provide a custom program name (in `PATH`) or a complete pathname to `pkg-config` which describes `CFLAGS`, `LIBS` and possibly other build-time options in `*.pc` files, to use third-party libraries. On build systems which support multiple architectures you may also want to set `PKG_CONFIG_PATH` to match your current build.

H.8.2 LibGD

```
--with-gd-includes="-I/foo/bar"
```

If you installed `libgd` in some place where your C preprocessor can't find the header files, use this switch to add additional `-I` flags.

```
--with-gd-libs="-L/foo/bar -labcd -lxyz"
```

If your copy of `libgd` isn't linking properly, use this to give the proper `-L` and `-l` flags to make it work. See `LIBS=` in `gd's` Makefile.

Note

the `--with-gd` switches are not necessary if you have `gd 2.0.8` or higher installed properly. The `gdlib-config` script or `pkg-config` manifest will be detected and used by default in that situation.

```
--with-gdlib-config
```

This option allows to provide a custom program name (in `PATH`) or a complete pathname to `gdlib-config`. This may be needed on build systems which support multiple architectures, or in cases where your distribution names this program differently.

H.8.3 LibUSB

```
--with-libusb-config
```

This option allows to provide a custom program name (in `PATH`) or a complete pathname to `libusb-config` (usually delivered only for `libusb-0.1` version, but not for `libusb-1.0`). This may be needed on build systems which support multiple architectures or provide several versions of `libusb`, or in cases where your distribution names this program differently.

H.8.4 Various

```
--with-ssl-includes, --with-usb-includes, --with-snmp-includes,  
--with-neon-includes, --with-libltdl-includes,  
--with-powerman-includes="-I/foo/bar"
```

If your system doesn't have `pkg-config` and support for any of the above libraries isn't found (but you know it is installed), you must specify the compiler flags that are needed.

```
--with-ssl-libs, --with-usb-libs, --with-snmp-libs,  
--with-neon-libs, --with-libltdl-libs  
--with-powerman-libs="-L/foo/bar -R/foo/bar -labcd -lxyz"
```

If system doesn't have `pkg-config` or it fails to provides hints for some of the settings that are needed to set it up properly and the build in defaults are not right, you can specify the correct values for your system here.

I Upgrading notes

This file lists changes that affect users who installed older versions of this software. When upgrading from an older version, be sure to check this file to see if you need to make changes to your system.

Note

For packaging (OS distribution or in-house) it is recommended to primarily `./configure --with-all` and then excise `--without-something` explicitly for items not supported on your platform, so you do not miss out on new NUT features as they come with new releases. Some may require that you update your build environment with new third-party dependencies, so a broken build of a new NUT release would let you know how to act.

This is a good time to point out that for stricter packaging systems, it may be beneficial to add `--enable-option-checking=fatal` to the `./configure` command line, in order to quickly pick up any other removed option flags.

I.1 Changes from 2.8.2 to 2.8.3

- NUT development snapshots can now have more version components than the standard semantic versioning triplet, optionally adding the amount of commits on the development trunk since previous release, and the amount of commits on a feature branch that are unique to it. Release artifacts that have zeroes in both positions would have them stripped and still have the standard "semver" exposed, but the development snapshots can now be more reasonably upgraded with automated tooling. A copy of the current version information would be embedded into "dist" archives as a `VERSION_DEFAULT` file, so it can be used without git. Certain distros can benefit from a `VERSION_FORCED` file or a `NUT_VERSION_FORCED` environment variable exported from their build system, e.g. via `echo NUT_VERSION_FORCED=1.1.1 > VERSION_FORCED`. Unfortunately, some appliances tag all software the same with their firmware version; if this is required, a `(NUT_)VERSION_FORCED_SEMVER` envvar or file can help identify the actual NUT release version triplet used on the box. Please use it, it immensely helps with community troubleshooting! Documentation about this would be maintained in `docs/nut-versioning.adoc` [[issue #1949](#)]
- When packaging or custom-building for (Linux) distributions with `systemd`, you can now take advantage of `nut-systemd.preset` file to enable or disable certain NUT units by default; its comments document each choice. [[issue #2721](#)]
- A `nut-udev-settle.service` was introduced to replace dependency on the `systemd-udev-settle.service` which is deprecated and causes warnings on some systems. It was shown to benefit NUT use-cases however. [[#2638](#)]
- Reference packages prepared by `make package` should now separate the `PACKAGE_VERSION` from the platform-dependent prefix by a dash character in the ultimate package file name. Previously they were glued together for some platform targets (HPUX, Solaris). Solaris SVR4 package file names should now differentiate `i386` vs. `amd64` and `sparc` vs. `sparcv9`, depending on `target_cpu` of the build. If you had any scripts relying on the older pattern, they may have to be updated.
- The `make dist` goal now takes more care to require availability of the man pages to put into the prepared distribution archive. Development and CI builds on platforms unable to fulfill this goal can use `make distcheck-ci` (and `make dist-ci`) to fake presence of pre-built man pages with placeholder files, to complete other aspects of `distcheck` validation. [[#2842](#)]
- the PyPI distribution of the `PyNUTClient` module tarball should now use a lower-cased file name (and immediate versioned directory name inside) to match the requirements of [PEP-0625](#). The Python module name (and its directory) should remain camel-cased. OS distribution package recipes that deliver the module separately (e.g. as part of Python ecosystem rather than NUT) may have to adjust. [[#2773](#)]
- The NUT-Monitor GUI application localization was updated; `make install` should now deliver also `xx_YY.UTF-8` pattern named symbolic links to the short-named directories and files involved, since some platforms insist on having those for translations to be found — this should be reflected in OS packaging recipes as well. [[#2845](#)]
- New `libupsclient` API methods added:
 - `upscli_str_add_unique_token()` and `upscli_str_contains_token()`, to help C NUT clients process `ups.status` and similarly structured strings same way as NUT core code base. [[#2852](#), [#2859](#)]
 - `upscli_set_default_connect_timeout()` to modify the internal timeout used by `upscli_connect()` (default 0 still means blocking connections, positive values should time-limit the connection attempts), and `upscli_get_default_` to retrieve its copy. [[#2847](#)]
- API versions of `libupsclient` and `libnutscan` export more symbols now, and so were bumped to new "current" numbers; this may impact the naming of shared object files to be delivered by updated packaging. [[#2895](#)]

- Standard NUT clients including `upsc`, `upscmd`, `upsrw`, `upslog`, `upsmon`, `upsimage`, `upsset` and `upsstats`) were updated to default with a 10-second connection establishment timeout in case of name resolution lags or unresponsive hosts (notably a problem with `upsmon` contacting many remote systems at once). This may potentially impact NUT deployments which somehow relied on the blocking behavior of these clients; you can use the `NUT_DEFAULT_CONNECT_TIMEOUT` environment variable to fix this. [[#2847](#)]
 - Several NUT clients including `upscmd`, `upsrw`, `upsimage`, `upsset`, `upsstats`, and `upslog` (during reconnection), did not `UPSCLI_CONN_TRYSSL` so went plaintext even when secure connections were possible. Fixed to at least try being secure, same way as `upsc` does for a long time. This may cause console or log messages when SSL can not be initialized, you can use the `NUT_QUIET_INIT_SSL` environment variable to suppress them where the cryptography is known to be not set up, so the warnings bring no value. [[#2847](#)]
 - `lib/*.pc.in`: propagate `-R/PATH` (or equivalent — as detected by the `configure` script for the currently used compiler and linker toolkits) in `pkg-config` metadata pointing to NUT library installation location (by default not in system prefix) to help third-party clients link with us automatically. If this causes issues, `--disable-ldflags-nut-rpath(-cxx)` options (or `--enable...="..."` with specific linker arguments) can help. [[#2782](#), [#2865](#)]
 - Updated man page generation with `configure` script options to specify that manual sections on the target platform differ from (Linux-based) defaults hard-coded into page sources; this should allow to simplify NUT packaging recipe maintenance in distributions (no more updating patches for changed or added documentation sources)
 - `upsmon` should now integrate natively with systemd-driven OS sleep events (built with systemd version 221 or newer "inhibitor interface"), so various hacks previously packaged into `/usr/lib/systemd/system-sleep/` scripts or units requiring/conflicting with the `sleep.target` may be obsolete. For fallback with older systemd, a `nut-sleep.service` is provided now. [[#1070](#), [#2596](#), [#2597](#)]
 - Added systemd and SMF service integration for `upslog` as a `nut-logger` service (disabled by default, needs a `upslog.conf` file to deliver the `UPSLOG_ARGS=...` setting for actual monitoring and logging). [[#1803](#)]
 - Handling of per-UPS ALARM state was introduced to `upsmon`, allowing it to optionally treat it as a factor in deciding that the device is in a "critical" state (polled more often, assumed dead if communications are lost). Since it is up to devices and their NUT drivers what they would raise as an alarm (might be something as mundane as ECO mode being active), some alarms can contribute to unwanted/early shutdowns. For this reason a `0|1` setting `ALARMCRITICAL` was introduced into `upsmon.conf` (default is 1), for such users to be able to prevent their `upsmon` from treating the ALARM status as overly severe when it is not in fact. [[#2658](#), [#415](#)]
 - `usbhid-ups` and `netxml-ups` updated to handle "No battery installed!" alarm also to set the RB (Replace Battery) value in `ups.status`. This may cause dual triggering of notifications (as an ALARM generally and as an important `REPLBATT` status in particular) in `upsmon`, but better safe than sorry. [[#415](#)]
 - `usbhid-ups` subdriver `PowerCOM` HID seemingly sent UPS shutdown and `stayoff` commands in wrong byte order, at least for devices currently in the field. Driver now sends the commands in a way that satisfies new devices; just in case a flag `toggle powercom_sdcmd_byte_order_fallback` was added to set the old behavior (if some devices do need it). [[PR #2480](#)]
 - `usbhid-ups` subdriver `CyberPower` HID default `pollfreq` sped up to 12 seconds (common default is 30 seconds). Feedback is welcome if this improves connection stability or overwhelms the UPS controller instead. [[issue #1689](#), [PR #2718](#)]
 - `usbhid-ups` subdriver `CyberPower` HID default `offdelay` is set to 60 and `ondelay` to 120 seconds, in accordance with man page suggestions; users with custom settings not divisible by 60 will be loudly warned. [[#1394](#)]
 - `snmp-ups` subdriver `netvision-mib`: synchronized `netvision_output_info` with the currently available `SOCOMEUPS-` this can impact some other devices using that MIB (negatively, if the older mappings were indeed correct for any practical cases, and were not a typo). [[#2803](#)]
 - `nutdrv_gx` fixed `hunnox_protocol()` to honour the optional `novendor` setting for devices that are confused by such query (e.g. DEXP LCD EURO 1200VA); it may be remotely possible that some other devices could begin to misbehave due to this fix — please let us know then. [[#2839](#)]
-

- mge-utalk driver will no longer set non-standard status values `COMMFALT` and `ALARM` (for a specific status bit); instead, it will set modern `ups.alarm` with values `COMMFALT` and/or `DEVICEALARM` (and raise an `ALARM` in `ups.status` for either, as standard alarms go). If your clients (e.g. custom parsing scripts) for devices supported by this driver depended on those non-standard tokens in `ups.status`, they would have to be updated to handle the new token values in `ups.alarm` instead. [#2708]
 - Added support for `lbrb_log_delay_sec=N` setting to delay propagation of `LB` or `LB+RB` state (buggy with APC BXnnnnMI devices/firmwares issued circa 2023-2024 which flood the logs with spurious `LOWBATT` and `REPLACEBATT` events). This may work better for some devices when combined with flags like `onlinedischarge_calibration` and `lbrb_log_delay_w` [#2347]
 - Enabled installation of built PDF and HTML (including man page renditions) files under the configured `docdir`. It seems previously they were only built (if requested) but not installed via `make`, unlike the common man pages which are delivered automatically. Packaging recipes can likely be simplified now. [#2445]
 - A `NUT_DEBUG_SYSLOG` environment variable was introduced to tweak activation of syslog message emission (and related detachment of `stderr` when daemons are backgrounding), which can be useful for systemd service units. It can be set via `nut.conf` file for all standard consumers, or patched/dropped-in to systemd unit definitions specifically (less recommended, but may be easier to package). The positive effect would be avoiding duplicate logging as both `syslog` and `stderr` ending up in the same journal. [#2394]
 - A `CHANGELOG_REQUIRE_GROUP_BY_DATE_AUTHOR` setting was added (for `make` calls and used by `tools/gitlog2chang` script), and it defaults to `true` allowing for better ordered documents at the cost of some memory during document generation. Resource-constrained builders (working from a Git workspace, not tarball archives) may have to set it to `false` when calling `make` for NUT. [#2510]
 - Drivers should now be able to set `STATEPATH` via `ups.conf` to match `upsd` custom configuration ability; in fact, the data server would prefer the value from `ups.conf` over the one in `upsd.conf`, if both are present. Note that `NUT_STATEPATH` environment variable trumps both. [issue #694]
 - NUT products like `nut-scanner`, which dynamically load shared libraries at run-time without persistent pre-linking, should now know the library file names that were present during build (likely encumbered with version suffixes), and prefer them over plain `libname.so` patterns used previously (which on some platforms are only delivered by development packages as symlinks). Packaging recipes can likely be simplified now: some distros certainly did patch NUT source to similar effect). [#2431]
 - Numerous changes to `nut-scanner` and symbols that its `libnutscan.so` delivers have caused a library version bump. New methods have been added and one structure (`nutscan_ipmi_t`) updated in a (hopefully) backwards compatible manner. [PR #2523, issue #2244 and numerous PRs for it]
 - The `nutconf` tool added to main codebase with NUT v2.8.2 release could be packaged as a single program (with just a dependency on `libnutscan`), e.g. the library code with configuration file processing logic was built into it. Starting with NUT v2.8.3, the `libnutconf` may optionally be built as a standalone shared library, to deliver for development of integrations using `--with-dev-libnutconf` option. In this case the `nutconf` tool program would also depend on it for run-time linking. This may have to be considered in packaging recipes. [#2828]
 - Internal API change for `sendsignalpid()` and `sendsignalfn()` methods, which can impact NUT forks which build using `libcommon.la` and similar libraries. Added new last argument with `const char *progrname` (may be `NULL`) to check that we are signalling an expected program name when we work with a PID. With the same effort, NUT programs which deal with PID files to send signals (`upsd`, `upsmon`, `drivers` and `upsdrcvtl`) would now default to a safety precaution — checking that the running process with that PID has the expected program name (on platforms where we can determine one). This might introduce regressions for heavily customized NUT builds (e.g. embedded in NAS or similar devices) whose binary file names differ significantly from a `progrname` defined in the respective NUT source file, so a boolean `NUT_IGNORE_CHECKPROCNAME` environment variable support was added to optionally disable this verification. Also the NUT daemons should request to double-check against their run-time process name (if it can be detected). [issue #2463]
 - More environment variable support was added to NUT programs, primarily aimed at wrappers such as init scripts and service unit definitions, allowing to tweak what (and whether) they write into debug traces, and so "make noise" or "bring invaluable insights" to logs or terminal; they can generally be used for services and init scripts via `nut.conf`:
 - See `NUT_IGNORE_CHECKPROCNAME` and `NUT_DEBUG_SYSLOG` above. [#1915]
-

- See `NUT_DEFAULT_CONNECT_TIMEOUT` above. [#2847]
- A `NUT_QUIET_INIT_BANNER` envvar (presence or "true" value) prevents tool name and NUT version banner from being printed out when programs start. [issues #1789 vs. #316]
- A configure script option to build `--with-modbus+usb` was added to let the caller insist on the use of USB-capable libmodbus (or fail the NUT build attempt). Certain build arguments can default this option to become enabled (implicitly): `configure --with-modbus --with-usb` and either `--with-drivers=*apc_modbus*` (actually implies `--with-modbus`) or `--with-modbus-includes=... --with-modbus-libs=...` as a way to avoid surprises with custom NUT builds aiming to have an USB-capable `apc_modbus` driver (currently this requires a custom-built libmodbus, can be a static build to avoid conflicts with OS). [#2666]
- A configure script option to `--enable-NUT_STRARG-always` was added to enable the `NUT_STRARG` macro (to handle NULL string printing) even if system libraries seem to safely support this behavior natively. This should primarily help against overly zealous static analysis tools in recent compiler generations. [#2585]

I.2 Changes from 2.8.1 to 2.8.2

- Builds requested with a specific C/C language standard revision via `CFLAGS` and `CXXFLAGS` should again be honoured. There was a mishap with the m4 scripting for `autoconf` which could have caused use of `C11/C11` if compiler supported it, regardless of a request. [PR #2306]
- Added generation of FreeBSD/pfSense quirks for USB devices supported by NUT (may get installed to `$datadir` e.g. `/usr/local/share/nut` and need to be pasted into your `/boot/loader.conf.local`). [#2159]
- `nut-scanner` now does not propose active bus, busport and device values when generating device configurations by default. They may appear as comments, or enabled by specifying the `-U` command-line option several times. [#2221]
- The `tools/gitlog2changelog.py.in` script was revised, in particular to convert section titles (with contributor names) into plain ASCII character set, for `dblatex` versions which do not allow diacritics and other kinds of non-trivial characters in sections. A number of other projects seem to use the NUT version of the script, and are encouraged to look at related changes in `configure.ac` and `Makefile.am` recipes. [PR #2360, PR #2366]

I.3 Changes from 2.8.0 to 2.8.1

- NUT documentation recipes were revised, so many of the text source files were renamed to `*.adoc` pattern. Newly, a `release-notes.pdf` and HTML equivalents are generated. Packages which deliver documentation may need to update the lists of files to ship. [#1953] Developers may be impacted by new `configure --enable-spellcheck` toggle (should add spelling checks to `make check` by default, if tools are available) to facilitate quicker acceptance of contributions. Packaging systems may now want to explicitly disable it, if it blocks package building (pull requests to update the `docs/nut.dict` are a better and welcome solution). [#2067]
- Several improvements regarding simultaneous support of USB devices that were previously deemed "identical" and so NUT driver instances did not start for all of them:
 - Some more drivers should now use the common USB device matching logic and the `7ups.conf` options for that [#1763], and man pages were updated to reflect that [#1766];
 - The `nut-scanner` tool should suggest these options in its generated device configuration [#1790]: hopefully these would now suffice for sufficiently unique combinations;
 - The `nut-scanner` tool should also suggest sanity-check violations as comments in its generated device configuration [#1810], e.g. bogus or duplicate serial number values;
 - The common USB matching logic was updated with an `allow_duplicates` flag (caveat emptor!) which may help monitor several related no-name devices on systems that do not discern "bus" and "device" values (although without knowing reliably which one is which... sometimes it is better than nothing) [#1756].

- Work on NUT for Windows branch led to situation-specific definitions of what in POSIX code was all "file descriptors" (an `int` type). Now such entities are named `TYPE_FD`, `TYPE_FD_SER` or `TYPE_FD_SOCKET` with some helper macros to name and determine "invalid" values (closed file, etc.) Some of these changes happened in NUT header files, and at this time it was not investigated whether the set of files delivered for third-party code integration (e.g. C/C++ projects binding with `libnutclient` or `libupsclient`) is consistent or requires additional definitions/files. If something gets broken by this, it is a bug to address in future [#1556]
 - Further revision of public headers delivered by NUT was done, particularly to address lack of common data types (`size_t`, `ssize_t`, `uint16_t`, `time_t` etc.) in third-party client code that earlier sufficed to only include NUT headers. Sort of regression by NUT 2.8.0 (note those consumers still have to re-declare some numeric variable types used) [#1638]
 - For practical example of NUT consumer adaptation (to cater to both old and new API types) please see <https://github.com/collectd/collectd/pull/4043>
 - Added support for `make install` of PyNUT module and NUT-Monitor desktop application—such activity was earlier done by packages directly; now the packaging recipes may use NUT source-code facilities and package just symlinks as relevant for each distro separately [#1462, #1504]
 - The `upsd.conf` listing of `LISTEN` addresses was previously inverted (the last listed address was applied first), which was counter-intuitive and fixed for this release. If user configurations somehow relied on this order (e.g. to prioritize IPv6 vs. IPv4 listeners), configuration changes may be needed. [#2012]
 - The `upsd` configured to listen on IPv6 addresses should handle only IPv6 (and not IPv4-mappings like it might have done before) to avoid surprises and insecurity—if user configurations somehow relied on this dual support, configuration changes may be needed to specify both desired IP addresses. Note that the daemon logs will now warn if a host name resolves to several addresses (and will only listen on the first hit, as it did before in such cases). [#2012]
 - A definitive behavior for `LISTEN *` directives became specified, to try handling both IPv4 and IPv6 "any" address (subject to `upsd` CLI options to only choose one, and to OS abilities). This use-case may be practically implemented as a single IPv6 socket on systems with enabled and required IPv4-mapped IPv6 address support, or as two separate listening sockets - logged messages to this effect (e.g. inability to listen on IPv4 after opening IPv6) are expected on some platforms. End-users may also want to reconfigure their `upsd.conf` files to remove some now-redundant `LISTEN` lines. [#2012]
 - Added support for `make sockdebug` for easier developer access to the tool; also if `configure --with-dev` is in effect, it would now be installed to the configured `libexec` location. A man page was also added. [#1936]
 - NUT software-only drivers (dummy-ups, clone, clone-outlet) separated from serial drivers in respective Makefile and configure script options - this may impact packaging decisions on some distributions going forward [#1446]
 - GPIO category of drivers was added (`--with-gpio` configure script option) - this may impact packaging decisions on some (currently Linux released 2018+) distributions going forward [#1855]
 - An explicit `configure --with-nut-scanner` toggle was added, specifically so that build environments requesting `--with-all` but lacking `libltdl` would abort and require the packager either to install the dependency or explicitly forfeit building the tool (some distro packages missed it quietly in the past) [#1560]
 - An `upsdebugx_report_search_paths()` method in NUT common code was added, and exposed in `libnutscan.so` builds in particular - API version for the public library was bumped [#317]
 - Some environment variable support was added to NUT programs, primarily aimed at wrappers such as init scripts and service unit definitions, allowing to tweak what (and whether) they write into debug traces, and so "make noise" or "bring invaluable insights" to logs or terminal:
 - A `NUT_DEBUG_LEVEL=NUM` envvar allows to temporarily boost debugging of many daemons (`upsd`, `upsmon`, `drivers`, `upsdvctl`, `upssched`) without changes to configuration files or scripted command lines. [#1915]
 - A `NUT_DEBUG_PID` envvar (presence) support was added to add current process ID to tags with debug-level identifiers. This may be useful when many NUT daemons write to the same console or log file, such as in containers/plugins for Home Assistant, storage appliances, etc. [#2118]
 - A `NUT_QUIET_INIT_SSL` envvar (presence or "true" value) prevents `libupsclient` consumers (notoriously `upsc`) from reporting whether they have initialized SSL support. [#1662]
-

- A `NUT_QUIET_INIT_UPSNOTIFY` envvar (presence or "true" value) prevents daemons which can notify service management frameworks (such as `systemd`) about passing their lifecycle milestones, to not report loudly if they could not do so (e.g. running on a system without a framework, or misconfigured so they could not report and the OS would restart the false-positively "unresponsive" service). [#2136]
- `configure` script, reference init-script and packaging templates updated to eradicate `@PIDPATH@/nut` ambiguity in favor of `@ALTPIDPATH@` for the unprivileged processes vs. `@PIDPATH@` for those running as root [#1719]
- The "layman report" of NUT configuration options displayed after the run of `configure` script can now be retained and installed by using the `--enable-keep_nut_report_feature` option; packagers are welcome to make use of this, to better keep track of their deliveries [#1826, #1708]
- Renamed generated `nut-common.tmpfiles(.in) ⇒ nut-common-tmpfiles.conf(.in)` to install a `/usr/lib/systemd-tmpfiles/*.conf` pattern [#1755]
 - If earlier NUT v2.8.0 package recipes for your Linux distribution dealt with this file, you may have to adjust its name for newer releases.
 - Several other issues have been fixed related to this file and its content, including #1030, #1037, #1117 and #1712
- Extended Linux `systemd` support with optional notifications about daemon state (READY, RELOADING, STOPPING) and `watchdog` keep-alive messages. Note that `WatchdogSec=` values are currently NOT pre-set into `systemd` unit file templates provided by NUT, this is an exercise for end-users based on sizing of their deployments and performance of monitoring station [#1590, #1777]
- `snmp-ups`: some subdrivers (addressed using the driver parameter `mibs`) were renamed: `pw` is now `eaton_pw_nm2`, and `pxgx-ups` is `eaton_pxgx-ups` [#1715]
- The `tools/gitlog2changelog.py.in` script was revised, in particular to generate the `ChangeLog` file more consistently with different versions of Python interpreter, and without breaking the long file paths in the resulting mark-up text. Due to this, a copy of this file distributed with NUT release archives is expected to considerably differ on first glance from its earlier released versions (not just adding lines for the new release, but changing lines in the older releases too) [#1945, #1955]

1.4 Changes from 2.7.4 to 2.8.0

- Note to distribution packagers: this version hopefully learns from many past mistakes, so many custom patches may be no longer needed. If some remain, please consider making pull requests for upstream NUT codebase to share the fixes consistently across the ecosystem. Also note that some new types of drivers (so package groups with unique dependencies) could have appeared since your packaging was written (e.g. with `modbus`), as well as new features in `systemd` integration (`nut-driver@instances` and the `nut-driver-enumerator` to manage their population), as well as updated Python 2 and Python 3 support (again, maybe dictating different package groups) as detailed below.
- Due to changes needed to resolve build warnings, mostly about mismatching data types for some variables, some structure definitions and API signatures of several routines had to be changed for argument types, return types, or both. Primarily this change concerns internal implementation details (may impact update of NUT forks with custom drivers using those), but a few changes also happened in header files installed for builds configured `--with-dev` and so may impact `upsc` and `nutclient` (C++) consumers. At the very least, binaries for those consumers should be rebuilt to remain stable with NUT 2.8.0 and not mismatch int-type sizes and other arguments.
- `libusb-1.0`: NUT now defaults to building against `libusb-1.0` API version if the `configure` script finds the development headers, falling back to `libusb-0.1` if not. Please report any regressions.
- `apcupsd-ups`: When monitoring a remote `apcupsd` server, interpret "SHUTTING DOWN" as a NUT "LB" status. If you were relying on the previous behavior (for instance, in a monitor-only situation), please adjust your `upsmon` settings. Reference: <https://github.com/networkupstools/nut/issues/460>
- Packagers: the `AsciiDoc` detection has been reworked to allow NUT to be built from source without requiring `asciidoc/a2x` (using pre-built man pages from the distribution tarball, for instance). Please double-check that we did not break anything (see `docs/configure.txt` for options).

- Driver core: options added for standalone mode (scanning for devices without requiring `ups.conf`) - see `docs/man/nutupsdrv.txt` for details.
 - `oldmge-shut` has been removed, and replaced by `mge-shut`.
 - New drivers for devices with "Qx" (also known as "Megatec Q*") family of protocols should be developed as sub-drivers in the `nutdrv_qx` framework for USB and Serial connected devices, not as updates/clones of older e.g. `blazer` family and `bestups`. Sources, man pages and start-up messages of such older drivers were marked with "OBSOLETION WARNING".
 - `liebert-esp2`: some multi-phase variable names have been updated to match the rest of NUT.
 - `netxml-ups`: if you have old firmware, or were relying on values being off by a factor of 10, consider the `do_convert_deci` flag. See `docs/man/netxml-ups.txt` for details.
 - `snmp-ups`: detection of Net-SNMP has been updated to use `pkg-config` by default (if present), rather than `net-snmp-config` (script(s) as the only option available previously. The scripts tend to specify a lot of options (sometimes platform-specific) in suggested `CFLAGS` and `LIBS` compared to the packaged `pkg-config` information which also works and is more portable. If this change bites your distribution, please bring it up in <https://github.com/networkupstools/nut/issues> or better yet, post a PR. Also note that `./configure --with-netsnmp-config(=yes)` should set up the preference of the detected script over `pkg-config` information, if both are available, and `--with-netsnmp-config=/path/name` would as well.
 - `snmp-ups`: bit mask values for flags in earlier codebase were defined in a way that caused logically different items to have same numeric values. This was fixed to surely use different definitions (so changing numbers behind some of those macro symbols), and testing with UPS, ePDU and ATS hardware which was available did not expose any practical differences.
 - `usbhid-ups`: numeric data conversion from wire protocol to CPU representation in `GetValue()` was completely reworked, aiming to be correct on all CPU types. That said, regressions are possible and feedback is welcome.
 - `nut-scanner`: Packagers, take note of the changes to the library search code in `common/common.c`. Please file an issue if this does not work with your platform.
 - `dummy-ups` can now specify `mode` as a driver argument, and separates the notion of `dummy-once` (new default for `*.dev` files that do not change) vs. `dummy-loop` (legacy default for `*.seq` and others) [[issue #1385](#)]
 - Note this can break third-party test scripts which expected `*.dev` files to work as a looping sequence with a `TIMER` keywords to change values slowly; now such files should get processed to the end once. Specify `mode=dummy-loop` driver option or rename the data file used in the `port` option for legacy behavior. Use/Test-cases which modified such files content externally should not be impacted.
 - Python: scripts have been updated to work with Python 3 as well as 2.
 - PyNUT module (protocol binding) supports both Python generations.
 - NUT-Monitor (desktop UI client) got separated into two projects: one with support for Python2 and GTK2, and another for Python3 and Qt5. On operating systems that serve both environments, either of these implementation should be usable. For distributions that deprecated and removed Python2 support, it is a point to consider in NUT packages and their build-time and installation dependencies. The historic filenames for desktop integration (`NUT-Monitor` script and `nut-monitor.desktop`) are still delivered, but now cover a wrapper script which detects the environment capabilities and launches the best suitable UI implementation (if both are available).
 - `apcsmart`: updates to CS "hack" (see `docs/man/apcsmart.txt` for details)
 - `upsdebugx()`: added `[D#]` prefix to log entries with level `> 0` so if any scripts or other tools relied on parsing those messages making some assumptions, they should be updated
 - `upsdebugx()` and related methods are now macros, optionally calling similarly named implementations like `s_upsdebugx()` as a slight optimization; this may show up in linking of binaries for some customized build scenarios
 - libraries, tools and protocol now support a `TRACKING ID` to be used with an `INSTCMD` or `SET VAR` requests; for details see `docs/net-protocol.txt` and `docs/sock-protocol.txt`
 - `upswr`: display the variable type beside `ENUM / RANGE`
 - Augeas: new `--with-augeas-lenses-dir` configure option.
-

I.5 Changes from 2.7.3 to 2.7.4

- `scripts/systemd/nut-server.service.in`: Restore systemd relationship since it was preventing `upsd` from starting whenever one or more drivers, among several, was failing to start
- Fix UPower device matching for recent kernels, since `hiddev*` devices now have class "usbmisc", rather than "usb"
- `macosx-ups`: the "port" driver option no longer has any effect
- Network protocol information: default to type `NUMBER` for variables that are not flagged as `STRING`. This point is subject to improvements or change in the next release 2.7.5. Refer to `docs/net-protocol.txt` for more information

I.6 Changes from 2.7.2 to 2.7.3

- The `nutdrv_qx(8)` driver will eventually supersede `bestups(8)`. It has been tested on a U-series Patriot Pro II. Please test the new driver on your hardware during your next maintenance window, and report any bugs.
- If you are upgrading from a new install of 2.7.1 or 2.7.2, double-check the value of `POWERDOWNFLAG` in `$prefix/etc/upsmon.conf` - it has been restored to `/etc/killpower` as in 2.6.5 and earlier.
- If you use `upslog` with a large sleep value, you may be interested in adding `killall -SIGUSR1 upslog` to any `OB/OL` script actions. This will force `upslog` to write a log entry to catch short power transients.
- Be sure that your SSL keys are readable by the NUT system user. The SSL subsystem is now initialized after `upsd` forks, to work around issues in the NSS library.
- The systemd `nut-server.service` does not Require `nut-driver` to be started successfully. This was previously preventing `upsd` startup, even for just one driver failure among many. This also matches the behavior of sysV `initscripts`.

I.7 Changes from 2.7.1 to 2.7.2

- `upsdrcctl` is now installed to `$prefix/sbin` rather than `$driverexec`. This usually means moving from `/bin` to `/sbin`, apart from few exceptions. In all cases, please adapt your scripts.
- FreeDesktop Hardware Abstraction Layer (HAL) support was removed. Please adapt your packaging files, if you used to distribute the `nut-hal-drivers` package.
- This is a good time to point out that for stricter packaging systems, it may be beneficial to add "`--enable-option-checking=fatal`" to the `./configure` command line, in order to quickly pick up any other removed option flags.

I.8 Changes from 2.6.5 to 2.7.1

- The `apcsmart(8)` driver has been replaced by a new implementation. There is a new parameter, `ttymode`, which may help if you have a non-standard serial port, or Windows. In case of issues with this new version, users can revert to `apcsmart-old`.
- The `nutdrv_qx(8)` driver will eventually supersede `blazer_ser` and `blazer_usb`. Options are not exactly the same, but are documented in the `nutdrv_qx` man page.
- Mozilla NSS support has been added. The OpenSSL configuration options should be unchanged, but please refer to the `upsd.conf(5)` and `upsmon.conf(5)` documentation in case we missed something.
- `upsrw(8)` now prints out the maximum size of variables. Hopefully you are not parsing the output of `upsrw` - it would be easier to use one of the NUT libraries, or implement the network protocol yourself.
- The `jNut` source is now here: <https://github.com/networkupstools/jNut>

I.9 Changes from 2.6.4 to 2.6.5

- users are encouraged to update to NUT 2.6.5, to fix a regression in upssched.
- mge-shut driver has been replaced by a new implementation (newmge-shut). In case of issue with this new version, users can revert to oldmge-shut. UPDATE: oldmge-shut was dropped between 2.7.4 and 2.8.0 releases.

I.10 Changes from 2.6.3 to 2.6.4

- users are encouraged to update to NUT 2.6.4, to fix upsd vulnerability (CVE-2012-2944: upsd can be remotely crashed).
- users of the bestups driver are encouraged to switch to blazer_ser, since bestups will soon be deprecated.

I.11 Changes from 2.6.2 to 2.6.3

- nothing that affects upgraded systems.

I.12 Changes from 2.6.1 to 2.6.2

- apcsmart driver has been replaced by a new implementation. In case of issue with this new version, users can revert to apcsmart-old.

I.13 Changes from 2.6.0 to 2.6.1

- nothing that affects upgraded systems.

I.14 Changes from 2.4.3 to 2.6.0

- users of the megatec and megatec_usb drivers must respectively switch to blazer_ser and blazer_usb.
- users of the liebertgxt2 driver are advised that the driver name has changed to liebert-esp2.

I.15 Changes from 2.4.2 to 2.4.3

- nothing that affects upgraded systems.

I.16 Changes from 2.4.1 to 2.4.2

- The default subdriver for the blazer_usb driver USB id 06da:0003 has changed. If you use such a device and it is no longer working with this driver, override the *subdriver* default in *ups.conf* (see man 8 blazer).
- NUT ACL and the allowfrom mechanism has been replaced in 2.4.0 by the LISTEN directive and tcp-wrappers respectively. This information was missing below, so a double note has been added.

I.17 Changes from 2.4.0 to 2.4.1

- nothing that affects upgraded systems.
-

I.18 Changes from 2.2.2 to 2.4.0

- The nut.conf file has been introduced to standardize startup configuration across the various systems.
- The cpsups and nitram drivers have been replaced by the powerpanel driver, and removed from the tree. The cyberpower driver may suffer the same in the future.
- The al175 and energizerups drivers have been removed from the tree, since these were tagged broken for a long time.
- Developers of external client application using libupsclient must rename their "UPSCONN" client structure to "UPSCONN_t".
- The upsd server will now disconnect clients that remain silent for more than 60 seconds.
- The files under scripts/python/client are distributed under GPL 3+, whereas the rest of the files are distributed under GPL 2+. Refer to COPYING for more information.
- The generated udev rules file has been renamed with dash only, no underscore anymore (i.e. 52-nut-usbups.rules instead of 52_nut-usbups.rules)

I.19 Changes from 2.2.1 to 2.2.2

- The configure option "--with-lib" has been replaced by "--with-dev". This enable the additional build and distribution of the static version of libupsclient, along with the pkg-config helper and manual pages. The default configure option is to distribute only the shared version of libupsclient. This can be overridden by using the "--disable-shared" configure option (distribute static only binaries).
- The UPS poweroff handling of the usbhid-ups driver has been reworked. Though regression is not expected, users of this driver are encouraged to test this feature by calling "upsmon -c fsd" and report any issue on the NUT mailing lists.

I.20 Changes from 2.2.0 to 2.2.1

- nothing that affects upgraded systems. (The below message is repeated due to previous omission)
- Developers of external client application using libupsclient are encouraged to rename their "UPSCONN" client structure to "UPSCONN_t" since the former will disappear by the release of NUT 2.4.

I.21 Changes from 2.0.5 to 2.2.0

- users of the newhidups driver are advised that the driver name has changed to usbhid-ups.
 - users of the hidups driver must switch to usbhid-ups.
 - users of the following drivers (powermust, blazer, fentonups, mustek, esupssmart, ippon, sms) must switch to megatec, which replaces all these drivers. Please refer to doc/megatec.txt for details.
 - users of the mge-shut driver are encouraged to test newmge-shut, which is an alternate driver scheduled to replace mge-shut,
 - users of the cpsups driver are encouraged to switch to powerpanel which is scheduled to replace cpsups,
 - packagers will have to rework the whole nut packaging due to the major changes in the build system (completely modified, and now using automake). Refer to packaging/debian/ for an example of migration.
 - specifying -a <id> is now mandatory when starting a driver manually, i.e. not using upsdvctl.
 - Developers of external client application using libupsclient are encouraged to rename the "UPSCONN" client structure to "UPSCONN_t" since the former will disappear by the release of NUT 2.4.
-

I.22 Changes from 2.0.4 to 2.0.5

- users of the newhidups driver: the driver is now more strict about refusing to connect to unknown devices. If your device was previously supported, but fails to be recognized now, add *productid=XXXX* to *ups.conf*. Please report the device to the NUT developer's mailing list.

I.23 Changes from 2.0.3 to 2.0.4

- nothing that affects upgraded systems.
- users of the following drivers (powermust, blazer, fentonups, mustek, esupssmart, ippon, sms, masterguard) are encouraged to switch to megatec, which should replace all these drivers by nut 2.2. For more information, please refer to *doc/megatec.txt*

I.24 Changes from 2.0.2 to 2.0.3

- nothing that affects upgraded systems.
- hidups users are encouraged to switch to newhidups, as hidups will be removed by nut 2.2.

I.25 Changes from 2.0.1 to 2.0.2

- The newhidups driver, which is the long run USB support approach, needs hotplug files installed to setup the right permissions on device file to operate. Check newhidups manual page for more information.

I.26 Changes from 2.0.0 to 2.0.1

- The cyberpower1100 driver is now called cpsups since it supports more than just one model. If you use this driver, be sure to remove the old binary and update your *ups.conf* *driver=* setting with the new name.
- The *upsstats.html* template page has been changed slightly to reflect better HTML compliance, so you may want to update your installed copy accordingly. If you've customized your file, don't just copy the new one over it, or your changes will be lost!

I.27 Changes from 1.4.0 to 2.0.0

- The sample config files are no longer installed by default. If you want to install them, use *make install-conf* for the main programs, and *make install-cgi-conf* for the CGI programs.
- ACCESS is no longer supported in *upsd.conf*. Use ACCEPT and REJECT.

– Old way:

```
ACCESS grant all adminbox
ACCESS grant all webserver
ACCESS deny all all
```

– New way:

```
ACCEPT adminbox
ACCEPT webserver
REJECT all
```

– Note that ACCEPT and REJECT can take multiple arguments, so this will also work:

```
ACCEPT adminbox webserver
REJECT all
```

- The drivers no longer support `sddelay` in `ups.conf` or `-d` on the command line. If you need a delay after calling `upsdrvctl shutdown`, add a call to `sleep` in your shutdown script.
- The templates used by `upsstats` have changed considerably to reflect the new variable names. If you use `upsstats`, you will need to install new copies or edit your existing files to use the new names.
- Nobody needed UDP mode, so it has been removed. The only users seemed to be a few people like me with ancient `asapm-ups` binaries. If you really want to run `asapm-ups` again, bug me for the new patch which makes it work with `upscient`.
- `make install-misc` is now `make install-lib`. The `misc` directory has been gone for a long time, and the target was ambiguous.
- The `newapc` driver has been renamed to `apcsmart`. If you previously used `newapc`, make sure you delete the old binary and fix your `ups.conf`. Otherwise, you may run the old driver from 1.4.

1.28 File trimmed here on changes from 1.2.2 to 1.4.0

For information before this point, start with version 2.4.1 and work back.

J Project history

This page is an attempt to document how everything came together.

The Network UPS Tools team would like to warmly thank Russell Kroll.

Russell initially started this project, maintaining and improving it for over 8 years (1996 — mid 2005).

J.1 Prototypes and experiments

J.1.1 May 1996: early status hacks

APC's Powerchute was running on `kadets.d20.co.edu` (a BSD/OS box) with SCO binary emulation. Early test versions ran in `cron`, pulled status from the log files and wrote them to a `.plan` file. You could see the results by fingering `pwrchute@kadets.d20.co.edu` while it lasted:

```
Last login Sat May 11 21:33 (MDT) on tttyp0 from intrepid.rmi.net
Plan:
Welcome to the UPS monitor service at kadets.d20.co.edu.
The Smart-UPS attached to kadets generated a report at 14:24:01 on 05/17/96.
During the measured period, the following data points were taken:
Voltage ranged from 115.0 VAC to 116.3 VAC.
The UPS generated 116.3 VAC at 60.00 Hz.
The battery level was at 27.60 volts.
The load placed on the UPS was 024.9 percent.
UPS temperature was measured at 045.0 degrees Celsius.
Measurements are taken every 10 minutes by the upsd daemon.
This report is generated by a script written by Russell Kroll<rkroll@kadets>.
Modified for compatibility with the BSD/OS cron daemon by Neil Schroeder
```

This same status data could also be seen with a web browser, since we had rigged up a CGI wrapper script which called `finger`.

J.1.2 January 1997: initial protocol tests

Initial tests with a freestanding non-daemon program provided a few basic status registers from the UPS. The 940-0024C cable was not yet understood, so this happened over the [attachment:apcevilhack.jpg evil two-wire serial hack].

```
Communicating with SMART-UPS 700 S/N WS9643050926 [10/17/96]
Input voltage range: 117.6 VAC - 118.9 VAC
Load is 010.9% of capacity, battery is charged to 100.0% of capacity
```

Note that today's `apcsmart` driver still displays the serial number when it starts, since it is derived from this original code.

J.1.3 September 1997: first client/server code

The first split daemon/client code was written. upsd spoke directly to the UPS (APC Smart models only) and communicated with upsc by sending binary structures in UDP datagrams.

The first CGI interface existed, but it was all implemented with shell scripts. The main script would call upsc to retrieve status values. Then it would cat a template file through sed to plug them into the page.



upsstats actually has since returned to using templates, despite having a period in the middle when it used hardcoded HTML.

The images were also created with shell scripts. Each script would call upsc to get the right value (utility, upload, battcap). It then took the value, plugged it into a command file with sed, and passed that into *fly*, a program which used an interpreted language to create images. *fly* actually uses *gd*, just like upsimage does today.

This code later evolved into Smart UPS Tools 0.10.

J.2 Smart UPS Tools

J.2.1 March 1998: first public release

Version 0.10 was released on March 10, 1998. It used the same design as the pre-release prototype. This made expansion difficult as the binary structure used for network communications would break any time a new variable was added. Due to byte-ordering and struct alignment issues, the code usually couldn't talk over the network to a system with a different architecture. It was also hopelessly bound to one type of UPS hardware.

Five more releases followed with this design followed. The last was 0.34, released October 27, 1998.

J.2.2 June 1999: Redesigned, rewritten

Following a long period of inactivity and two months of prerelease testing versions, 0.40.0 was released on June 5, 1999. It featured a complete redesign and rewrite of all of the code. The layering was now in three pieces, with the single driver (smartups) separate from the server (upsd).

Clients remained separate as before and still used UDP to talk to the server, but they now used a text-based protocol instead of the brittle binary structs. A typical request like "REQ UTILITY" would be answered with "ANS UTILITY 120.0".

The ups-trust425-625 driver appeared shortly after the release of 0.40.0, marking the first expansion beyond APC hardware.

Over the months that followed, the backupsprio driver would be forked from the smartups driver to handle the APC Back-UPS Pro line. Then the backups driver was written to handle the APC Back-UPS contact-closure models. These drivers would later be renamed and recombined, with smartups and backupsprio becoming apcsmart, and backups became genericups.

The drivers stored status data in an array. At first, they passed this data to upsd by saving it to a file. upsd would reread this file every few seconds to keep a copy for itself. This was later expanded to allow shared memory mode, where only a stub would remain on the disk. The drivers and server then passed data through the shared memory space.

upsd picked up the ability to monitor multiple drivers on the system, and the "upsname@hostname" scheme was born. Access controls were added, and then the network code was expanded to allow TCP communications, which at this point were on port 3305.

J.3 Network UPS Tools

J.3.1 September 1999: new name, new URL

Several visitors to the web page and subscribers to the mailing lists provided suggestions to rename the project. The old name no longer accurately described it, and it was perilously close to APC's "Smart-UPS" trademark. Rather than risk problems in the future, the name was changed. Kern Sibbald provided the winner: Network UPS Tools, which captures the essence of the project and makes for great short tarball filenames: nut-x.y.z.tar.gz.

The new name was first applied to 0.42.0, released October 31, 1999. This is also when the web pages moved from the old <http://www.exploits.org/~rkroll/smartupstools/> URL to the replacement at <http://www.exploits.org/nut/> to coincide with the name change.

More drivers were written and the hardware support continued to grow. upsmon picked up the concepts of what is now known as "primary" and "secondary", and could now handle environments where multiple systems get power from a single UPS.

Manager mode was added to allow changing the value of read/write variables in certain UPS models.

J.3.2 June 2001: common driver core

Up to this point, all of the drivers compiled into freestanding programs, each providing their own implementation of main(). This meant they all had to check the incoming arguments and act uniformly. Unfortunately, not all of the programs behaved the same way, and it was hard to document and use consistently. It also meant that startup scripts had to be edited depending on what kind of hardware was attached.

Starting in 0.45.0, released June 11, 2001, there was a new common core for all drivers called `main.c`. It provided the main function and called back to the `upsdrv_*` functions provided by the hardware-specific part of the drivers. This allowed driver authors to focus on the UPS hardware without worrying about the housekeeping stuff that needs to happen.

This new design provided an obvious way to configure drivers from one file, and so `ups.conf` was born. This eventually spawned `upsdrvctl`, and now all drivers based on this common core could be started or stopped with one command. Startup scripts now could contain "upsdrvctl start", and it didn't matter what kind of hardware or how many UPSes you had on one system.

Interestingly, at the end of this month, Arnaud Quette entered the UPS world, as a subcontractor of the now defunct MGE UPS SYSTEMS. This marked the start of a future successful collaboration.

J.3.3 May 2002: casting off old drivers, IANA port, towards 1.0

During the 0.45.x series, both the old standalone drivers and the ones which had been converted to the common core were released together. Before the release of 0.50.0 on May 24, 2002, all of the old drivers were removed. While this shrank the list of supported hardware, it set the precedent for removing code which isn't receiving regular maintenance. The assumption is that the code will be brought back up to date by someone if they actually need it. Otherwise, it's just dead weight in the tree.

This change meant that all remaining drivers could be controlled with the `upsdrvctl` and `ups.conf`, allowing the documentation to be greatly simplified. There was no longer any reason to say "do this, unless you have this driver, then do this".

IANA granted an official port number to the project, and the network code switched to port 3493. It had previously been on 3305 which is assigned to `odette-ftp`. 3305 was probably picked in 1997 because it was the fifth project to spawn from some common UDP server code.

After 0.50.1, the 0.99 tree was created to provide a tree which would receive nothing but bug fixes in preparation for the release of 1.0. As it turned out, very few things required fixing, and there were only three releases in this tree.

J.4 Leaving 0.x territory

J.4.1 August 2002: first stable tree: NUT 1.0.0

After nearly 5 years of having a 0.x version number, 1.0.0 was released on August 19, 2002. This milestone meant that all of the base features that you would expect to find were intact: good hardware support, a network server with security controls, and system shutdowns that worked.

The design was showing signs of wear from the rapid expansion, but this was intentionally ignored for the moment. The focus was on getting a good version out that would provide a reasonable base while the design issues could be addressed in the future, and I'm confident that we succeeded.

J.4.2 November 2002: second stable tree: NUT 1.2.0

One day after the release of 1.0.0, 1.1.0 started the new development tree. During that development cycle, the CGI programs were rewritten to use template files instead of hard-coded HTML, thus bringing back the flexibility of the original unreleased prototype from 5 years before. The `multimon` was removed from the tree, as the new `upsstats` could do both jobs by loading different templates.

A new client library called `upsclient` was created, and it replaced `upsfetch`. This new library only supported TCP connections, and used an opaque context struct to keep state for each connection. As a result, client programs could now do things that used multiple connections without any conflicts. This was done primarily to allow OpenSSL support, but there were other benefits from the redesign.

`upsd` and the clients could now use OpenSSL for basic authentication and encryption, but this was not included by default. This was provided as a bonus feature for those users who cared to read about it and enable the option, as the initial setup was complex.

After the 1.1 tree was frozen and deemed complete, it became the second stable tree with the release of 1.2.0 on November 5, 2002.

J.4.3 April 2003: new naming scheme, better driver glue, and an overhauled protocol

Following an extended period with no development tree, 1.3.0 got things moving again on April 13, 2003. The focus of this tree was to rewrite the driver-server communication layer and replace the static naming scheme for variables and commands.

Up to this point, all variables had names like `STATUS`, `UTILITY`, and `OUTVOLT`. They had been created as drivers were added to the tree, and there was little consistency. For example, it probably should have been `INVOLT` and `OUTVOLT`, but there was no `OUTVOLT` originally, so `UTILITY` was all we had. This same pattern repeated with `ACFREQ` — is it incoming or outgoing? — and many more.

To solve this problem, all variables and commands were renamed to a hierarchical scheme that had obvious grouping. `STATUS` became `ups.status`. `UTILITY` turned into `input.voltage`, and `OUTVOLT` is `output.voltage`. `ACFREQ` is `input.frequency`, and the new `output.frequency` is also now supported. Every other variable or command was renamed in this fashion.

These variables had been shared between the drivers and `upsd` as values. That is, for each name like `STATUS`, there was a `#define` somewhere in the tree with an `INFO_` prefix that gave it a number. `INFO_STATUS` was `0x0006`, `INFO_UTILITY` was `0x0004`, and so on, with each name having a matching number. This number was stored in an `int` within a structure which was part of the array that was either written to disk or shared memory.

That structure had several restrictions on expansion and was dropped as the data sharing method between the drivers and the server. It was replaced by a new system of text-based messages over Unix domain sockets. Drivers now accepted a short list of commands from upsd, and would push out updates asynchronously. upsd no longer had to poll the state files or shared memory. It could just select all of the driver and client fds and act on events.

At the same time, the network protocol on port 3493 was overhauled to take advantage of the new naming scheme. The existing "REQ STATUS@su700", "ANS STATUS@su700 OL" scheme was showing signs of age, and it really only supported the UPS name (@su700) as an afterthought. The new protocol would now use commands like GET and LIST, leading to exchanges like "GET VAR su700 ups.status" and "VAR su700 ups.status OL". These responses contain enough data to stand alone, so clients can now handle them asynchronously.

J.4.4 July 2003: third stable tree: NUT 1.4.0

On July 25, 2003, 1.4.0 was released. It contained support for both the old "REQ" style protocol (with names like STATUS), and the new "GET" style protocol (with names like ups.status). This tree is provided to bridge the gap between all of the old releases and the upcoming 2.0.

2.0 will be released without support for the old REQ/STATUS protocol. The hope is that client authors and those who have implemented their own monitoring software will use the 1.4 cycle to change to the new protocol. The 1.4 releases contain a lot of compatibility code to make sure both work at the same time.

J.4.5 July 2003: pushing towards 2.0

1.5.0 forked from 1.4.0 and was released on July 29, 2003. The first changes were to throw out anything which was providing compatibility with the older versions of the software. This means that 1.5 and the eventual 2.0 will not talk to anything older than 1.4.

This tree continues to evolve with new serial routines for the drivers which are intended to replace the aging upscommon code which dates back to the early 0.x releases. The original routines would call alarm and read in a tight loop while fetching characters. The new functions are much cleaner, and wait for data with select. This makes for much cleaner code and easier strace/ktrace logs, since the number of syscalls has been greatly reduced.

There has also been a push to make sure the data from the UPS is well-formed and is actually usable before sending updates out to upsd. This started during 1.3 as drivers were adapted to use the dstate functions and the new variable/command names. Some drivers which were not converted to the new naming scheme or didn't do sanity checks on the incoming UPS data from the serial port were dropped from the tree.

This tree was released as 2.0.0.

J.5 Backwards and Forwards Compatibility (NUT v1.x vs. v2.x)

The old network code spans a range from about 0.41.1 when TCP support was introduced up to the recent 1.4 series. It used variable names like STATUS, UTILITY, and LOADPCT. Many of these names go back to the earliest prototypes of this software from 1997. At that point there was no way to know that so many drivers would come along and introduce so many new variables and commands. The resulting mess grew out of control over the years.

During the 1.3 development cycle, all variables and instant commands were renamed to fit into a tree-like structure. There are major groups, like input, output and battery. Members of those groups have been arranged to make sense - input.voltage and output.voltage compliment each other. The old names were UTILITY and OUTVOLT. The benefits in this change are obvious.

The 1.4 clients can talk to either type of server, and can handle either naming scheme. 1.4 servers have a compatibility mode where they can answer queries for both names, even though the drivers are internally using the new format.

When 1.4 clients talk to 1.4 or 2.0 (or more recent) servers, they will use the new names.

Here's a table to make it easier to visualize:

	Server version			
Client version	1.0	1.2	1.4	2.0+

	Server version			
1.0	yes	yes	yes	no
1.2	yes	yes	yes	no
1.4	yes	yes	yes	yes
2.0+	no	no	yes	yes

Version 2.0, and more recent, do not contain backwards compatibility for the old protocol and variable/command names. As a result, 2.0 clients can't talk to anything older than a 1.4 server. If you ask a 2.0 client to fetch "STATUS", it will fail. You'll have to ask for "ups.status" instead.

Authors of separate monitoring programs should have used the 1.4 series to write support for the new variables and command names. Client software can easily support both versions as long as they like. If upsd returns *ERR UNKNOWN-COMMAND* to a GET request, you need to use REQ.

J.6 networkupstools.org

J.6.1 November 2003: a new URL

The bandwidth demands of a project like this have slowly been forcing me to offload certain parts to other servers. The download links have pointed offsite for many months, and other large things like certain UPS protocols have followed. As the traffic grows, it's clear that having the project attached to exploits.org is not going to work.

The solution was to register a new domain and set up mirrors. There are two initial web servers, with more on the way. The main project URL has changed from <http://www.exploits.org/nut/> to <https://www.networkupstools.org>. The actual content is hosted on various mirrors which are updated regularly with rsync, so the days of dribbling bits through my DSL should be over.

This is also when all of the web pages were redesigned to have a simpler look with fewer links on the left side. The old web pages used to have 30 or more links on the top page, and most of them vanished when you dropped down one level. The links are now constant on the entire site, and the old links now live in their own groups in separate directories.

J.7 Second major version

J.7.1 March 2004: NUT 2.0.0

NUT 2.0.0 arrived on March 23, 2004. The jump to version 2 shows the difference in the protocols and naming that happened during the 1.3 and 1.5 development series. 2.0 no longer ships with backwards compatibility code, so it's smaller and cleaner than 1.4.

J.8 The change of leadership

J.8.1 February 2005: NUT 2.0.1

The year 2004 was marked by a release slowdown, since Russell was busy with personal subjects. But the patches queue was still growing quickly.

At that time, the development process was still centralized. There was no revision control system (like the current Subversion repository), nor trackers to interact with NUT development. Russell was receiving all the patches and requests, and doing all the work on his own, including releases.

Russell was more and more thinking about giving the project leadership to Arnaud Quette, which finally happened with the 2.0.1 release in February 2005.

This marked a new era for NUT...

First, Arnaud aimed at opening up the development by creating a project on the [Debian Alioth Forge](#). This allowed to build the team of hackers that Russell dreamed about. It also allows to ensure NUT's continuation, whatever happens to the leader. And that would most of all boost the projects contributions.