

NUT Packager and Integrators Guide

Arnaud Quette

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
2.8.5.1685 2.8.5.1711- 1711+g1c4ab78c6	2026-10-30	Current release snapshot of Network UPS Tools (NUT).	
2.8.5	2026-04-07	Some changes to docs and recipes, especially for parallel builds; introduced a way to build with private shared libraries to save space. Updated NUT SEMVER definition some more, added a normalization and comparison script. Some rounds of code-hardening project. Added reference RPM/DEB recipes to test NUT iterations on OBS. Numerous driver updates, some new ones introduced.	JK
2.8.4	2025-08-07	Fixed a few regressions introduced by release v2.8.3. Some changes to docs and recipes, especially for parallel builds. Updated NUT SEMVER definition some more. Some rounds of code-hardening project. Numerous driver updates, some new ones introduced.	JK
2.8.3	2025-04-07	Some changes to docs and recipes, libupsclient 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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
2.8.0	2022-04-26	Change of maintainer. Many changes to API, docs (both style and content), and recipes, with a stress on non-regression test-ability, run-time debug-ability, general codebase maintainability, as well as OS integrations (notably nut-driver-enumerator for systemd and SMF service instance maintenance). Added a lot in area of CI support and documented pre-requisite package lists for numerous platforms, and CI agent set-up. Added libusb-1.x support and many new driver categories (and drivers), and daisychain device connection support. Instant commands enhanced with TRACKING to enable protocol-based waiting for completion of a particular INSTCMD or SET operation.	JK
2.7.4	2016-03-09	NUT variables namespace updated, in particular for outlet groups, alarms and thresholds, ATS devices, and battery.charger.status to supersede CHRG and DISCHRG flags published in ups.status readings. NUT network protocol extended with NUMBER type; some API changes.	AQ
2.7.3	2015-04-22	Documentation revised, including some API changes. Added NUT DDL links. NUT variables namespace updated.	AQ
2.7.2	2014-04-17	The nut-website project was offloaded into a separate repository. FreeDesktop HAL support was removed (obsoleted in GNOME consumer). Introduced nutdrv_atcl_usb driver.	AQ
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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
2.6.5	2012-08-08	New macosx-ups driver, new implementation of mge-shut driver. NUT variables namespace updated. Docs cleaned up and revised.	AQ
2.6.4	2012-05-31	New NUT network protocol commands (LIST CLIENTS, LIST RANGE and NETVER), and socket protocol commands (ADDRANGE, DELRANGE). NUT variables namespace updated. Introduced nut-recorder tool.	AQ
2.6.3	2012-01-04	No substantial changes to documentation.	AQ
2.6.2	2011-09-15	Introduced nut-scanner tool and nut-ipmipsu driver, systemd support, and a new apcsmart implementation.	AQ
2.6.1	2011-06-01	Introduced default.* and override.* optional settings in ups.conf, an ups.efficiency report, and outlet.0 special handling.	AQ
2.6.0	2011-01-14	First release of AsciiDoc documentation for Network UPS Tools (NUT).	AQ

Contents

1	Introduction	1
2	Packagers involved	1
3	Possible use cases	2
4	Package boundaries	2
5	Build and installation policy	5
6	Service and shutdown integration	6
7	Packaging validation	6

Abstract

The aim of this document is to describe the best way to package the Network UPS Tools, and the best practices across the various packaging implementation of NUT.

So as to these can be spread on all supported platforms as a standard, and as a foundation block to build upon.

**Warning**

This is a Work In Progress document.

1 Introduction

Packaging is a final aim for software. It eases and completes the software integration into an OS, and allows users to have an easy software installation and support out of the box.

Note

Making NUT packaging more uniform should help its documentation, support and maintenance across the supported OSes.

2 Packagers involved

The following packagers are working on this subject:

- Debian (and derivatives): Arnaud Quette <aquette@debian.org>
- SUSE/Novell: Stanislav Brabec <sbrabec@suse.cz>
- Solaris, OpenSolaris, OpenIndiana and related illumos distributions: Jim Klimov <jimklimov@gmail.com>
- MacOS: Charles Lepple

Note

The people below should be contacted to (re)launch discussions!

The following packagers should be interested in working on this subject:

- FreeBSD: Thierry Thomas? <>
 - Mandriva: Oden Erikson? <>
 - RedHat / Fedora Core: <>
 - Gentoo: <>
 - NetBSD: <>
 - OpenBSD: <>
 - PLD: Andrzej Zawadzki <zawadaa@wp.pl>
 - E-Smith: Charlie Brady <charlieb-nut-upsdev@e-smith.com>
 - Windows: check with WinNUT author?!
 - HP-UX: <>
 - IBM AIX: <>
-

3 Possible use cases

A package selection should support the following deployments:

- "Standalone" for Local monitoring and shutdown: one host runs the driver(s), `upsd` and `upsmon` client for its own shutdown.
- Network data server: a host runs drivers and `upsd`, serving local and remote clients; `upsmon` is also needed if that host requires automatic shutdown.
- Network monitoring client: a host runs `upsmon` against a remote `upsd` and does not need local drivers or a local data server.
- Query and administration client: a host uses tools such as `upsc`, `upscmd` and `upsrw` to access a remote server without configuring automatic shutdown.
- Application integration: another program uses NUT client libraries or language bindings to communicate with an existing server.

4 Package boundaries

It is recommended to avoid creating a single package for all NUT features, installation of which may cause a huge footprint by both NUT programs and their third-party dependencies. Each end-user system has a different set of needs based on the devices they need to talk to and the user interface they prefer.

The aim of the package split suggested below is to:

- rationalize the split according to the above use cases;
- share resources used by many packages (descriptions, `i18n`, ...);
- find the best compromise between available features and dependencies: many features and drivers may involve significant dependency tree footprints, but are not of interest to each and every NUT user;
- standardize NUT package names across different operating systems;
- create the foundation for the upcoming and underway improvements;
- improve NUT integration;
- ease and improve user experience.

This standard was created by:

- capitalizing on the experience of existing packages;
- using and improving the use of all NUT features;
- considering upcoming NUT changes and improvements;
- working closely with packagers.

Package names, library ABI suffixes and dependency names vary by operating system. The following groups describe useful boundaries rather than a mandatory set of package names. A distribution may combine groups where that matches its packaging policy.

Many packaging ecosystems support short and long descriptions of each package. * The short description is used in the package manager's list of installed packages, and should be concise and informative. It is recommended to use a common format like `Network UPS Tools (NUT) - Group Name` (with the "Group Name" suffix taken from the table below) for this. * The long description is used in the package manager's detailed view of the package, and should be more verbose and descriptive. The initial paragraph common to all NUT packages should be:

Network UPS Tools (NUT) is a client/server monitoring system that allows computers to share uninterruptible power supply (UPS) and power distribution unit (PDU) hardware. Clients access the hardware through the server, and are notified whenever the power status changes.

Group and suggested package name	Purpose and contents	Dependency considerations
Umbrella package <code>nut</code>	Provide what the end-users commonly require to get started and can further append specific packages to.	At least <code>nut-common</code> , <code>nut-server</code> , <code>nut-clients</code> , <code>nut-scanner</code> ; maybe <code>nut-drivers-usb</code> as this is a common use case nowadays. If the packaging platform supports recommended dependencies based on what else is installed, the presence of X11 or similar environment is a good reason to include the <code>NUT-Monitor</code> and/or sibling <code>WMNut</code> GUI packages. Likewise, the presence of Augemas Tools package is a good reason to include <code>nut-augeas</code> .
Common integration <code>nut-common</code>	Shared configuration examples, account setup and service integration used by the selected NUT components. Contents are platform-dependent, e.g. <code>systemd-tmpfiles</code> , maybe <code>systemd-presets</code> , in case of modern Linux packaging. If you build NUT with separated private libraries which other binaries depend on for shared code, this is a good place to provide them, although you may want to split the <code>nut-common</code> package into <code>nut-common-config</code> (to keep it tiny) and <code>nut-common-lib</code> .	Keep ownership, permissions and runtime-directory creation consistent across packages; avoid installing the same shared file in multiple packages. Depended-on by most other NUT packages.
Data server and basic drivers <code>nut-server</code>	<code>upsd</code> and selected local drivers (software-only <code>dummy-ups</code> and <code>clone*</code>), with their configuration examples and manual pages. The majority of drivers may be split into further packages.	Include or depend on the libraries required by the drivers actually shipped. On MacOS, the <code>macosx-ups</code> driver may be part of this package.
Monitoring and command-line clients <code>nut-client-tools</code>	<code>upsmon</code> , <code>upssched</code> , <code>upsc</code> , <code>upscmd</code> , <code>upsrw</code> and <code>upslog</code> , with their configuration examples and manual pages.	A remote monitoring client should not require local hardware drivers. Include (depend on) the NUT client runtime library and selected SSL backend as applicable. On space-constrained systems, consider providing the <code>upsmon</code> and <code>upssched</code> programs as a separate package for just the graceful shutdown support.

Group and suggested package name	Purpose and contents	Dependency considerations
Client libraries <code>nut-client-lib</code> or OS-dictated, e.g. for C: <code>libupsclient1</code> for C++: <code>libnutclient1</code>	Runtime C and, when built, C++ client libraries.	Follow the platform's shared-library ABI naming and dependency policy; derive the package name from the library built for this release. The number in the package name, where required, is the latest compatible ABI version ("major" minus "age").
Development files <code>nut-client-lib-dev</code> or OS-dictated, e.g. for C: <code>libupsclient1-dev</code> for C++: <code>libnutclient1-dev</code>	Installed headers, linker metadata and <code>pkg-config</code> files for the libraries included in the build.	Depend on the corresponding runtime libraries and follow platform policy for development-package naming. Package name suffixes like <code>-dev</code> or <code>-devel</code> may be dictated by the OS.
CGI interface <code>nut-cgi</code> or <code>nut-client-cgi</code>	The built CGI programs, templates, configuration examples and manuals.	Include their linked libraries; web-server configuration remains a separate integration task and depends on the web-server of choice.
Device discovery <code>nut-scanner</code>	<code>nut-scanner</code> , <code>libnutscan</code> , and the installed development files and manuals selected for this build.	When dynamic library loading is enabled, distinguish direct linked dependencies from optional libraries needed for individual scan types. May be further separated into <code>nut-scanner-tools</code> and <code>nut-scanner-lib</code> (or <code>libnutscan1</code>) and corresponding development package. Note that the <code>nut-scanner</code> program and <code>libnutscan</code> are designed to avoid persistent linking to any external libraries that they may use to discover devices, so dependencies on e.g. <code>libsnmp</code> , <code>libneon</code> and <code>libusb</code> (0.1 or 1.0 depending on build-time choices and used by drivers) may be softly recommended but not required. If end-users are not interested in particular media/protocol as not used by their devices, they are not required to waste space on corresponding libraries.

Group and suggested package name	Purpose and contents	Dependency considerations
Optional driver families <code>nut-drivers-usb</code> <code>nut-drivers-snmpp</code> <code>nut-drivers-xml</code> <code>nut-drivers-ipmi</code> <code>nut-drivers-gpio</code> <code>nut-drivers-i2c</code> <code>nut-drivers-upower</code> <code>nut-drivers-powerman</code> <code>nut-drivers-modbus</code> etc.	Drivers using SNMP, NetXML, IPMI, Modbus, PowerMan, GPIO, I2C, UPower or platform-specific interfaces, as supported by the selected build. See the <code>configure</code> documentation or run-time help for details relevant to the NUT version you are packaging, and keep track of <code>UPGRADING.adoc</code> for changes in the driver population between releases.	Split by the dependencies or platform requirements of the actual driver binaries rather than assuming every host needs every family. Note that while serial port support is commonly available on all known platforms (does not require extra libraries on the target system), it may be still prudent to split the serial-only drivers into a separate <code>nut-serial</code> package rather than include them in the default footprint. Also note that some drivers support mixed dependencies, e.g. common <code>libmodbus</code> may use TCP and Serial, but a fork maintained by NUT (upstream pending) can also use USB. Some other drivers may be built to support both USB and Serial connections.
Python interface and client <code>nut-pynut</code> and <code>nut-monitor</code> or OS-dictated e.g. <code>python-pynut</code> and <code>python-nut-client</code>	PyNUTClient module and, when selected, the Python GUI NUT-Monitor.	Keep the module available independently of the graphical application, as it may have third-party consumers. Some packagers prefer to rely on the PyPI published package rather than building the module from NUT sources. The client module should be compatible with Python 2.7 and 3.x, so you may package it for each interpreter version. There are separate implementations of NUT-Monitor for Python 2.7 and 3.x however, with the former further supporting Gtk2, and the latter Qt5 or Qt6, as the GUI toolkit. A wrapper script takes care of launching the correct implementation, based on detected run-time support.
Documentation <code>nut-doc</code>	Generated user, developer, packaging and QA guides, and manual pages as appropriate for the documentation policy of the target distribution.	Decide whether individual manuals travel with their programs or with a documentation package, and avoid duplicate file ownership. This depends on the documentation policy of the target distribution, and technologies in their packaging systems (e.g. "facets" to optionally install some files).

When documenting package sizes, measure installed and archive sizes for the release, architecture and build configuration concerned. State that context alongside the measurements, since sizes vary between builds.

5 Build and installation policy

Select build options explicitly and record the resulting configuration summary. For packaging, `./configure --with-all` can help expose missing prerequisites and newly available features; disable unsupported features explicitly. `--with-all=auto`

instead selects features whose prerequisites are detected in the build environment, so the resulting package contents can change when that environment changes.

Use the options documented in `docs/configure.txt` to set installation paths. In particular, `--with-confdir` controls NUT's configuration location; options for driver, state, PID, CGI and HTML paths serve different purposes and should match the integration policy for the target distribution. Do not copy a path layout from another platform without checking its conventions.

Stage installation into a packaging directory with `DESTDIR`, as described in `INSTALL.nut.adoc`. Keep the configured final paths distinct from this staging prefix. Inspect the staged file inventory and derive package contents and dependencies from the actual build, including shared-library metadata and optional scanner libraries. Do not replace administrator configuration with example files during an upgrade.

6 Service and shutdown integration

Include the service definitions and shutdown hooks appropriate for the platform. Ensure that the selected driver, server and monitoring services use the same configuration and runtime paths as the built programs. `nut.conf` describes common deployment modes, but support for its settings varies between platform packages and service scripts; see [nut.conf\(5\)](#).

Keep normal service stop and machine shutdown handling distinct. Review how the installed shutdown hook uses the power-down flag and invokes the local driver shutdown sequence. This integration needs validation on the target platform; a successful build alone does not validate UPS power-off and restart behaviour.

7 Packaging validation

Validate the release archive with the appropriate `distcheck` target, then check the resulting package's staged contents, ownership, dependency metadata and upgrade handling. Verify service startup and shutdown integration in a suitable test deployment for the target platform. Document which optional features were included and where their configuration and manual pages are installed.
