NUT

An Introduction to Network UPS Tools

Configuration Examples

Based on

Network UPS Tools Project 2.8.0

Russell Kroll, Arnaud Quette, Jim Klimov,
Arjen de Korte, Charles Lepple and many others

Conforms to

RFC 9271 Uninterruptible Power Supply (UPS)
Management Protocol – Commands and Responses

Roger Price (Editor)

Version 3.0, with corrections up to 2023-03-20

This introduction is based on the Network UPS Tools (NUT) User Manual, the man pages and the file `config-notes.txt` which do not carry explicit copyright notices, but which are part of the NUT package which is GPL licensed.

Copyright © Russell Kroll, Arnaud Quette, Arjen de Korte, Charles Lepple and others

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
http://www.fsf.org/licenses/old-licenses/gpl-2.0.html

The User Manual provides the following notice:

> B. Acknowledgments / 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. This software would not be possible without their help.

Additional material:
Copyright © Roger Price 2017, 2018, 2020, 2021, 2022, 2023
Distributed under the GPLv3. http://www.fsf.org/licenses/gpl.html

---

The source file for this document has been marked up by the editor in LaTeX 2ε and rendered as PDF file ConfigExamples.A5.pdf in a portrait A5 format, 131 pages with one page per sheet. Your PDF viewer may be able to place two pages side by side on your big monitor.

The document is not only linear reading, but also hypertext. All chapters in the table of contents, all chapter references, all line number references throughout the document, all man page names and URL's are clickable. Such links are outlined in colour, for example `man ups.conf`. *If your mouse hovers over a clickable surface, your browser/PDF reader may tell you where the link leads.*

Page dimensions			
Dimension	**Design (A5)**	**Actual pt**	**Actual mm**
\hoffset	-29.4mm	-83.65106pt	-29.39963 mm
\voffset	-29.4mm	-83.65106pt	-29.39963 mm
\pdfpageheight	240mm	682.86613pt	239.99718 mm
\pdfpagewidth	197.5mm	561.94193pt	197.49768 mm
\textheight	210mm	597.50787pt	209.99753 mm
\textwidth	177.5mm	505.03642pt	177.49791 mm
\linewidth		505.03642pt	177.49791 mm
\columnsep	15mm	42.67912pt	14.99982 mm
\LinePrinterwidth	145.5mm	413.9876pt	145.49829 mm

Changes:

- 2017-06-27 First edition
- 2017-07-02 Added subsection "Configuration file formats". Added `lowbatt` to `ups.conf`. Added subsection "Driver daemon" to introduction. Added Ubuntu specific addresses.
- 2017-07-24 Added discussion of selective UPS shutdown to chapter 9.
- 2017-08-10 Added appendix D, "Using notify-send".
- 2018-01-10 Rewrote appendix D, "Using notify-send". Rewrote appendix A "Starting NUT". Added chapter 6.6 "For paranoïd sysadmins".
- 2018-08-22 In chapter 3.1 added reference to issue #597 for multiple UPS units.
- 2019-07-21 Added chapter 11 "Encrypted connections".
- 2020-08-20 File `heartbeat.dev` becomes `heartbeat.conf`
- 2020-09-30 Added Part 2 covering the Python3 scripts. Deprecated 11 "Encrypted connections".
- 2021-05-16 Split Part 2 into two parts: new Part 2 for the shim daemons, and a new part 3 for the Python3 replacement for upsmon and upssched. The Appendix becomes Part 4.
- 2021-06-06 Migrated figures from xfig to inkscape.
- 2021-08-03 Clarified that command `upsmon -c fsd` calls the command specified by declaration `SHUTDOWNCMD`.
- 2022-08-02 Updated to NUT software version 2.8.0, protocol version 1.3 and RFC 9271. Removed Part 3 UPSmon.py. Appendices become Part 3.
- 2022-11-27 Minor corrections.
- 2023-01-02 Passwords should not contain spaces or quotation marks ".
- 2023-03-20Rendering of the current document in its binary form.

Contents

1 UPS monitoring using NUT 1

1 Introduction, and Welcome to NUT 1
- 1.1 What is NUT? . . . 1
  - 1.1.1 NUT is a mature project . . . 2
- 1.2 You need to configure the NUT software . . . 2
- 1.3 Attachment Daemon upsd . . . 2
  - 1.3.1 Driver daemon . . . 4
- 1.4 Management Daemon upsmon . . . 4
  - 1.4.1 Utility program upsc . . . 5
- 1.5 Configuration file formats . . . 6
  - 1.5.1 Line spanning . . . 7
- 1.6 Mailing list: nut-users . . . 8
- 1.7 NUT has an RFC . . . 8

2 Simple server with no local users 9
- 2.1 Configuration file `ups.conf`, first attempt . . . 9
- 2.2 Configuration file `upsd.conf` . . . 10
- 2.3 Configuration file `upsd.users` . . . 10
- 2.4 Configuration file `upsmon.conf` for a simple server . . . 11
- 2.5 The delayed UPS shutdown . . . 14
- 2.6 The shutdown story for a simple server . . . 15
- 2.7 Configuration file `ups.conf` for a simple server, improved . . . 17
- 2.8 The shutdown story with quick power return . . . 17
- 2.9 Utility program upscmd . . . 18
- 2.10 Utility program upsrw . . . 18

3 Server with multiple power supplies 20
- 3.1 Configuration file `ups.conf` for multiple power supplies . . . 20
- 3.2 Configuration file `upsmon.conf` for multiple power supplies . . . 21
- 3.3 Shutdown conditions for multiple power supplies . . . 22

4 Workstation with local users 25
- 4.1 Configuration file `upsmon.conf` for a workstation . . . 26
- 4.2 Configuration file `upssched.conf` for a workstation . . . 28
  - 4.2.1 The `AT` declaration. . . . 28
- 4.3 Configuration script upssched-cmd for a workstation . . . 29

4.4 The shutdown story for a workstation . . . 31

5 Workstations share a UPS 32
5.1 Configuration file upsmon.conf for a secondary . . . 33
5.2 Configuration file upssched.conf for a secondary . . . 35
5.3 Configuration script upssched-cmd for a secondary . . . 36
5.4 NUMATTACH: Counting the protected systems . . . 37
5.5 Magic: How does the primary shut down the secondaries? . . . 37

6 Workstation with heartbeat 38
6.1 Configuration file ups.conf for workstation with heartbeat . . . 39
6.2 Configuration file heartbeat.conf for workstation . . . 40
6.3 Configuration file upsmon.conf for workstation with heartbeat . . . 40
6.4 Configuration file upssched.conf for workstation with heartbeat . . . 41
6.5 Script upssched-cmd for workstation with heartbeat . . . 41
6.6 For paranoïd sysadmins . . . 43

7 Workstation with timed shutdown 44
7.1 Configuration file ups.conf for workstation with timed shutdown . . . 45
7.2 Configuration file heartbeat.conf for workstation with timed shutdown . . . 46
7.3 Configuration file upsd.conf with timed shutdown . . . 46
7.4 Configuration file upsd.users with timed shutdown . . . 47
7.5 Configuration file upsmon.conf with timed shutdown . . . 47
7.6 Configuration file upssched.conf with timed shutdown . . . 50
7.6.1 The AT declaration . . . 51
7.7 Script upssched-cmd for workstation with timed shutdown . . . 52
7.7.1 The timed shutdown . . . 53
7.8 The timed shutdown story . . . 54

8 Workstation with additional equipment 56
8.1 Configuration files nut.conf . . . 57
8.2 Configuration files ups.conf and heartbeat.conf . . . 58
8.3 Configuration files upsd.conf . . . 59
8.4 Configuration files upsd.users . . . 59
8.5 Configuration file upsmon.conf . . . 60
8.6 Configuration file upssched.conf for mgmt . . . 63
8.6.1 UPS-3 on gold . . . 63
8.6.2 UPS-2 on gold . . . 64
8.6.3 UPS-1 on mgmt . . . 65
8.6.4 heartbeat on mgmt . . . 65
8.7 User script upssched-cmd . . . 65
8.8 The shutdown story . . . 68

2 TLS support for upsd and clients 69

9 Introduction 69
9.1 Use of Python3 . . . 69
9.1.1 No object orientation . . . 69
9.1.2 Lint-free code . . . 69

10 mkNUTcert.py builds TLS certificates 71
10.1 Very Short Introduction to TLS Certificates . . . 71
10.2 Overview of mkNUTcert.py . . . 73
10.3 What mkNUTcert.py provides . . . 74
10.3.1 Private Key and Certificate = Root Certificate . . . 74
10.3.2 Public Key Certificate . . . 75
10.4 Running mkNUTcert.py . . . 77

11 Encrypted connections 78
11.1 Additional configuration files . . . 79
11.1.1 In the remote server "gold" . . . 79
11.1.2 In each management client "mgmt" . . . 79
11.2 Debugging: Sniffing port 3493 . . . 80
11.3 Testing the TLS setup . . . 81

12 Shim daemons upsdTLS.py and upsmonTLS.py 82
12.1 Overview of Shim upsdTLS.py . . . 82
12.2 Overview of Shim upsmonTLS.py . . . 84
12.3 Summary of shims upsdTLS.py and upsmonTLS.py . . . 86
12.4 Running the shims upsdTLS.py and upsmonTLS.py . . . 86
12.4.1 Enabling the shims upsdTLS.py and upsmonTLS.py . . . 88
12.4.2 Listing the systemd activity . . . 89

3 Appendices 90

A Starting NUT 90

B Stopping NUT 92
B.1 Delayed UPS shutdown with NUT script . . . 92
B.2 Delayed UPS shutdown with a systemd service unit . . . 93

C Users and Directories for NUT 94

D Using notify-send **96**
D.1 What's wrong with notify-send? . . . 96
D.2 Give user "nut" ("upsd") the right to act as any user . . . 97
D.3 Search for and notify logged in users . . . 98
D.4 Testing the notify-send-all setup . . . 98
D.5 References for notify-send . . . 99

E Log rotation for upsdTLS.py and UPSmon.py **100**

4 UPS monitoring using Python3 script **101**

F Python3 script UPSmon.py version 1.2 **101**
F.1 What is UPSmon.py ? . . . 101
F.1.1 Principal differences between upsmon and UPSmon.py . . . 102
F.2 Compatibility with upsmon. . . . 104
F.3 Overview of UPSmon.py . . . 104
F.4 Running UPSmon.py . . . 106
F.5 UPSmon.py's events based un upsd's status changes . . . 108
F.5.1 UPSmon.py's additional status symbols and events . . . 109
F.6 Configuration file . . . 110
F.6.1 Initial declarations . . . 111
F.6.2 Group declarations . . . 111

G Typing alternative text bracketing characters **115**

H Grammar for `UPSmon.conf` **116**
H.1 Lexical structure . . . 116
H.2 Yacc Grammar . . . 118

I UPSmon.py configuration **121**
I.1 Configuration tool mkUPSmonconf.py version 1.3 . . . 121
I.2 `UPSmon.conf` configuration examples . . . 123
I.2.1 Timed shutdown plan, part 1 of 4, the introduction . . . 123
I.2.2 Timed shutdown plan, part 2 of 4, the shutdown . . . 124
I.2.3 Timed shutdown plan, part 3 of 4, warnings . . . 125
I.2.4 Timed shutdown plan, part 4 of 4, heartbeat . . . 127
I.2.5 Standard shutdown plan . . . 128
I.3 Redundant power supplies . . . 129
I.3.1 `MINSUPPLIES` failure: Timed shutdown plan . . . 129
I.3.2 `MINSUPPLIES` failure: Standard shutdown plan . . . 129

J UPSmon.py installation checklist **130**

5 The End **131**

K Acknowledgments **131**

L Errors, omissions, obscurities, confusions, typpos... **131**

List of Figures

1 Overview of NUT. 1
2 Symbols used in ups.status maintained by upsd. 3
3 Wall power has failed. 4
4 Symbols used to represent NOTIFY events maintained by upsmon. 5
5 NUT terms changed by NUT 2.8.0 and the RFC. 8
6 Server with no local users. 9
7 Configuration file ups.conf, first attempt. 9
8 Configuration file upsd.conf. 10
9 Configuration file upsd.users for a simple server. 10
10 Configuration file upsmon.conf for a simple server, part 1 of 5. 11
11 Configuration file upsmon.conf for a simple server, part 2 of 5. 11
12 Configuration file upsmon.conf for a simple server, part 3 of 5. 12
13 Configuration file upsmon.conf for a simple server, part 4 of 5. 12
14 Flags declaring what upsmon is to do for NOTIFY events. 12
15 Configuration file upsmon.conf for a simple server, part 5 of 5. 12
16 Delayed UPS shutdown. 14
17 openSUSE script for delayed UPS shutdown. 14
18 Configuration file ups.conf, improved. 17
19 Server with multiple power supplies. 20
20 File ups.conf for multiple power supplies. 21
21 Configuration file upsmon.conf for multiple power supplies, part 1 of 5. 21
22 Experiment to show effect of lost UPS. Part 1, ... 22
23 Experiment to show effect of lost UPS. Part 2, ... 23
24 Workstation with local users. 25
25 Configuration file upsmon.conf for a workstation, part 1 of 5. 26
26 Configuration file upsmon.conf for a workstation, part 2 of 5. 26
27 Configuration file upsmon.conf for a workstation, part 3 of 5. 27
28 Configuration file upsmon.conf for a workstation, part 4 of 5. 27
29 Configuration file upsmon.conf for a workstation, part 5 of 5. 27

30 Configuration file upssched.conf for a Debian 11 workstation, Part 1. 28
31 Configuration file upssched.conf for a Debian 11 workstation, Part 2. 28
32 Configuration script upssched-cmd for a workstation. 29
33 “Secondary” workstations take power from same UPS as “primary”. 32
34 Configuration file upsmon.conf for a secondary, part 1 of 5. 33
35 Configuration file upsmon.conf for a secondary, part 2 of 5. 33
36 Configuration file upsmon.conf for a secondary, part 3 of 5. 34
37 Configuration file upsmon.conf for a secondary, part 4 of 5. 34
38 Configuration file upsmon.conf for a secondary, part 5 of 5. 35
39 Configuration file upssched.conf for a secondary. 35
40 Configuration script upssched-cmd for a secondary. 36
41 Workstation with heartbeat. 38
42 Configuration file ups.conf for workstation with heartbeat. 39
43 Configuration file heartbeat.conf for workstation. 40
44 Configuration file upsmon.conf for a workstation with heartbeat. 40
45 Configuration file upssched.conf for a workstation with heartbeat. 41
46 Configuration script upssched-cmd including heartbeat. 42
47 Heartbeat watcher. 43
48 Workstation with timed shutdown. 44
49 Configuration file ups.conf for workstation with timed shutdown. 45
50 This is the configuration file heartbeat.conf for a workstation with timed shutdown. 46
51 Configuration file upsd.conf for workstation with timed shutdown. 46
52 This is the configuration file upsd.users for a simple server. 47
53 Configuration file upsmon.conf with timed shutdown, part 1 of 5. 47
54 Configuration file upsmon.conf with timed shutdown, part 2 of 5. 48
55 Configuration file upsmon.conf with timed shutdown, part 3 of 5. 49
56 Configuration file upsmon.conf with timed shutdown, part 4 of 5. 49
57 Configuration file upsmon.conf with timed shutdown, part 5 of 5. 50
58 Configuration file upssched.conf with timed shutdown, part 1. 50
59 Configuration file upssched.conf with timed shutdown, part 2. 51
60 Configuration script upssched-cmd for timed shutdown, 1 of 2. 52
61 Configuration script upssched-cmd for timed shutdown, 2 of 2. 53
62 Workstation with additional equipment. 56
63 File nut.conf for gold. 57
64 Files nut.conf for mgmt. 57
65 File ups.conf for gold. 58
66 File ups.conf for mgmt. 58
67 heartbeat.conf for mgmt. 58
68 File upsd.conf for gold. 59
69 File upsd.conf for mgmt. 59
70 File upsd.users for gold. 59

71 File upsd.users for mgmt . . . 59
72 Configuration file upsmon.conf for mgmt, part 1 of 5. . . . 60
73 Configuration file upsmon.conf for mgmt, part 2 of 5. . . . 61
74 Configuration file upsmon.conf for mgmt, part 3 of 5. . . . 62
75 Configuration file upsmon.conf for mgmt, part 4 of 5. . . . 62
76 Configuration file upsmon.conf for mgmt, part 5 of 5. . . . 63
77 Configuration file upssched.conf for mgmt . . . 64
78 User script upssched-cmd on mgmt, 1 of 5. . . . 65
79 User script upssched-cmd on mgmt, 2 of 5. . . . 66
80 User script upssched-cmd on mgmt, 3 of 5. . . . 66
81 User script upssched-cmd on mgmt, 4 of 5. . . . 67
82 User script upssched-cmd on mgmt, 5 of 5. . . . 67
83 File pylintrc, Changes to the default Python style. . . . 70
84 Command mkNETcert.py --help. . . . 73
85 Root certificate = private key and certificate. . . . 74
86 The client's PEM encoded public certificate. . . . 75
87 The self-signed public certificate. . . . 76
88 Encrypted connection to remote server. . . . 78
89 tcpdump of systemctl start nut-monitor.service without encryption. . . . 80
90 Unencrypted traffic on port 3493 (nut). . . . 81
91 Encrypted traffic on port 401. . . . 81
92 NUT 2.7.4 TLS support using shims upsdTLS.py and upsmonTLS.py. . . . 82
93 Command upsdTLS.py --help . . . 83
94 Command upsmonTLS.py --help . . . 84
95 Summary of upsdTLS.py and upsmonTLS.py options and default values. . . . 86
96 systemd service unit nut-py-server-shim.service for upsdTLS.py. . . . 87
97 systemd service unit nut-py-client-shim.service for upsmonTLS.py. . . . 87
98 Example of systemd service unit activity for NUT. . . . 89
99 Configuration file nut.conf. . . . 90
100 Daemons used by NUT. . . . 90
101 UPS shutdown script nutshutdown. . . . 92
102 UPS shutdown script nutshutdown for 2 of 3 UPS's. . . . 92
103 UPS shutdown service unit nut-delayed-ups-shutdown.service. . . . 93
104 Users and directories for NUT. . . . 94
105 Example of a notification. . . . 96
106 Modifications to file /etc/sudoers . . . 97
107 Bash script notify-send-all . . . 98
108 Log rotation for upsdTLS.py and UPSmon.py . . . 100
109 UPSmon.py requires TLS. . . . 101
110 Actions provided by UPSmon.py. . . . 103
111 % substitutions available in messages. . . . 103

112 Command `UPSmon.py --help` . . . 104
113 systemd service unit `nut-py-monitor.service` for UPSmon.py. . . . 106
114 Symbols used to represent events monitored by UPSmon.py. . . . 108
115 Additional status symbols generated by UPSmon.py. . . . 109
116 Additional events monitored by UPSmon.py. . . . 109
117 System log urgency levels. . . . 114
118 Alternative text bracketing characters. . . . 115
119 `UPSmon.conf` lexer tokens. . . . 117
120 `UPSmon.conf` lexer tokens. . . . 117
121 Representation of grammar production 118
122 `UPSmon.conf` grammar. . . . 118
123 `UPSmon.conf` grammar, continued. . . . 119
124 `UPSmon.conf` grammar, final part. . . . 120
125 Command `mkUPSmonconf.py --help` . . . 121
126 Timed shutdown plan, part 1 of 4, the introduction. . . . 123
127 Timed shutdown plan, part 2 of 4, the shutdown. . . . 124
128 Timed shutdown plan, part 3 of 4, warnings, . . . 126
129 Configuration file `heartbeat.conf` . . . 127
130 Addition to the file `ups.conf` for `heartbeat.conf` . . . 127
131 Timed shutdown plan, part 4 of 4, heartbeat. . . . 128
132 Standard shutdown plan differences . . . 128
133 Timed shutdown on `MINSUPPLIES` failure . . . 129
134 upsd and UPSmon.py runtime processes . . . 130

Part 1

UPS monitoring using NUT

The first part of this documentation discusses UPS activity monitoring using the facilities provided by NUT 2.8.0. Part 2 discusses TLS support for upsd and the clients. Part 3 provides technical appendices.

1 Introduction, and Welcome to NUT

1.1 What is NUT?

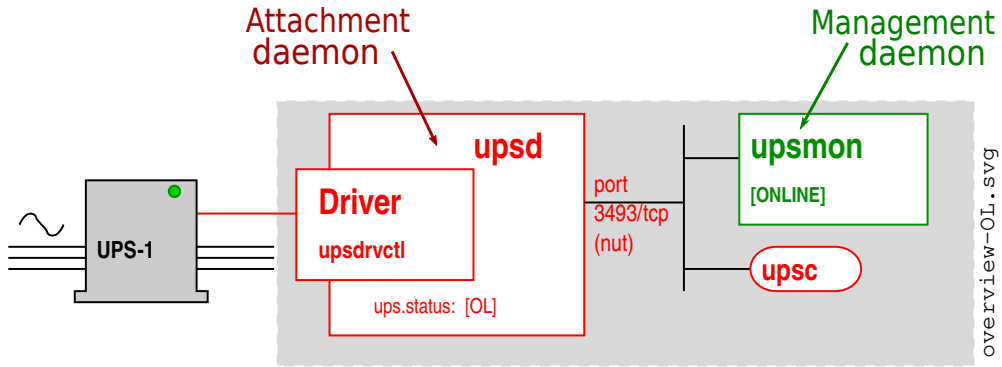


Figure 1: Overview of NUT.

The acronym NUT stands for "Network UPS Tools". It is a collection of GPL licensed software written in K&R style C for managing power devices, mainly UPS units. It supports a wide range of UPS units and can handle one or multiple UPS's of different models and manufacturers simultaneously in home, small business and larger professional installations. NUT replaces the software which came with your UPS.

The NUT software is included as a package in most major distributions of Linux, and the source code is available in a tarball for the others.

The NUT software includes complete technical documentation in the form of PDF manuals, configuration notes such as file `config-notes.txt`, man pages, a web site http://networkupstools.org and detailed comments in the sample configuration files supplied with the project. There is also a FAQ on the project web site, and a `nut-upsuser` mailing list in which users may ask questions.

1.1.1 NUT is a mature project

NUT was already operating in its current form when it registered port 3493/TCP (nut) with IANA in May 2002. Since then, the project has kept its principal characteristics which are the basis of its success:

- *Simplicity* – The design of NUT is simple and straightforward. No additional tools or software systems are needed to encode the messages sent between the attachment daemon and the management daemon.
- *Resilience* – The ability to operate in a *challenged environment*. Such environments are now receiving attention, for example the evolving vocabulary provided by RFC 7228 Terminology for Constrained-Node Networks.
- *Aggregation* – The simultaneous handling of a wide variety of UPS-things with widely differing capabilites. This is now known as the "Internet of Things", and again is the subject of much attention.

1.2 You need to configure the NUT software

To make full use of your UPS you will need to configure the NUT software used to manage UPS units. The technically complete documentation does not provide many examples; this introduction is intended to fill the gap by providing fully worked examples for some frequently met configurations. It is aimed at experienced Unix/Linux system administrators who are new to NUT. Pick the configuration which corresponds most closely to your installation, get it working, and then adapt it to your needs. If you have questions for the mailing list it is much easier to explain what you are trying to do by referring to a well known example.

1.3 Attachment Daemon upsd

Figure 1 shows the basic components of the NUT software. upsd is a daemon which runs permanently in the box to which one or more UPS's are attached. It scans the UPS's through the UPS-specific driver[1] and maintains an abstracted image of the UPS in memory[2].

The various parts of the abstracted image have standardized names, and a key part is the variable `ups.status` which gives the current status of the UPS unit. The current status is a string of symbols. The principal symbols are shown in figure 2, but if you write software which processes upsd symbols, expect to find other values in exceptional UPS specific cases.

Some important status values are [OL] which means that the UPS unit is taking power from the wall, and [OB LB] which means that wall power has failed, the UPS is supplying power from it's battery, and that battery is almost exhausted.

[1] See the Hardware Compatibility list and required drivers at https://www.networkupstools.org/stable-hcl.html
[2] This image may be viewed at any time with the command `upsc name-of-UPS`

Daemon upsd listens on port 3493/tcp (nut) for requests from its clients, which may be local or remote. It is amusing to test this using a tool such as `nc` or `netcat` and a UPS called `UPS-1`.

```
1 rprice@maria:~> REQUEST="GET VAR UPS-1 battery.charge"
2 rprice@maria:~> echo $REQUEST | nc localhost 3493
3 VAR UPS-1 battery.charge "100"
```

Chapter 1.4.1 will show that this is best done with NUT utility program upsc.

Later chapters will discuss the configuration files `ups.conf`, `upsd.conf` and `upsd.users` with the specific examples. For gory details, read `man upsd`, `man upsd.conf`, `man upsd.users` and `man ups.conf`.

[OL]	UPS unit is receiving power from the wall.
[OB]	UPS unit is not receiving power from the wall and is using its own battery to power the protected device.
[LB]	The battery charge is below a critical level specified by the variable `battery.charge.low`.
[RB]	UPS battery needs replacing.
[CHRG]	The UPS battery is currently being charged.
[DISCHRG]	The UPS battery is not being charged and is discharging.
[ALARM]	An alarm situation has been detected in the UPS unit.
[OVER]	The UPS unit is overloaded.
[TRIM]	The UPS voltage trimming is in operation.
[BOOST]	The UPS voltage boosting is in operation.
[BYPASS]	The UPS unit is in bypass mode.
[OFF]	The UPS unit is off.
[CAL]	The UPS unit is being calibrated.
[TEST]	UPS test in progress.
[FSD]	Tell secondary upsmon instances that final shutdown is underway.

Figure 2: Symbols used in `ups.status` maintained by upsd.

1.3.1 Driver daemon

The driver is a daemon which is part of the attachment daemon[3]. It talks to the UPS hardware and is aware of the state of the UPS. One of the strengths of the NUT project is that it provides drivers for a wide range of UPS units from a range of manufacturers. NUT groups the UPS's into families with similar interfaces, and supports the families with drivers which match the manufacturer's interface. See the hardware compatibility list for a looong list of the available drivers.

The drivers share a command interface, upsdrvctl, which makes it possible to send a command to the UPS without having to know the details of the UPS protocol. We will see this command in action in chapter 2.5 when we need to shut down the UPS after a system shutdown.

1.4 Management Daemon upsmon

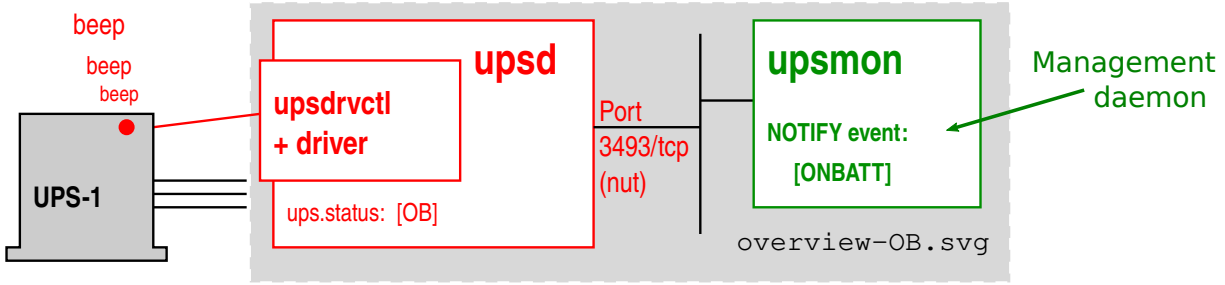


Figure 3: Wall power has failed.

The management daemon upsmon is an example of a client of upsd. It runs permanently as a daemon in a local or remote box, polling the status changes of the UPS unit. It is able to react to changes in the UPS state for example by emitting warning messages, or shutting down the box. The actions are specified in the configuration file `upsmon.conf` which will be discussed in specific examples.

As the state of a UPS evolves, the key status changes, called "NOTIFY events", are identified with the symbols shown in figure 4. The NOTIFY event symbol is also known as a "notifytype" in NUT.

Figure 3 shows what happens when wall power fails. Daemon upsd has polled the UPS, and has discovered that the UPS is supplying power from it's battery. The `ups.status` changes to [OB].

[3]Communication between upsd and each driver is through a socket which Debian 11 declares in directory `/var/run/nut` . The following example shows the sockets to two drivers `usbhid-ups` and `dummy-ups` :

```
root@titan   ls -alF /run/nut
drwxrwx-- 2 root nut 140 Aug 7 15:57 ./
drwxr-xr-x 30 root root 880 Aug 7 16:01 ../
srw-rw-- 1 nut nut 0 Aug 7 15:57 dummy-ups-heartbeat=
-rw-r-r- 1 nut nut 5 Aug 7 15:57 dummy-ups-heartbeat.pid
-rw-r-r- 1 nut nut 5 Aug 7 15:57 upsd.pid
srw-rw-- 1 nut nut 0 Aug 7 15:57 usbhid-ups-Eaton=
-rw-r-r- 1 nut nut 4 Aug 7 15:57 usbhid-ups-Eaton.pid
```

NOTIFY events based on status changes	
[ONLINE]	Status change [OB]→[OL]. The UPS is back on line.
[ONBATT]	Status change [OL]→[OB]. The UPS is now on battery.
[LOWBATT]	Status [LB] has appeared. The driver says the UPS battery is low.
[REPLBATT]	The UPS needs to have its battery replaced. Not all UPS's can indicate this.
NOTIFY events based on upsmon activity	
[FSD]	No status change. The primary has commanded the UPS into the "forced shutdown" mode.
[SHUTDOWN]	The local system is being shut down.
[COMMOK]	Communication with the UPS has been established.
[COMMBAD]	Communication with the UPS was just lost.
[NOCOMM]	The UPS can't be contacted for monitoring.
NOTIFY event based on NUT process error	
[NOPARENT]	upsmon parent died - shutdown impossible.

Figure 4: Symbols used to represent NOTIFY events maintained by upsmon.

Daemon upsmon has polled upsd, has discovered the status change and has generated the NOTIFY event [ONBATT].

For the gory details, read `man upsmon` and `man upsmon.conf`.

1.4.1 Utility program upsc

The NUT project provides this simple utility program to talk to upsd and retrieve details of the UPS's. For example, "What UPS's are attached to the local host?"

```
4  rprice@maria:~> upsc -L
5  UPS-1: Example Mfg ASR 1500 USBS
6  heartbeat: Heart beat validation of NUT
```

Let's ask for the upsd abstracted image of a UPS:

```
7  rprice@maria:~> upsc UPS-1
8  battery.charge: 100
9  battery.charge.low: 50
10  ...
11 driver.name: usbhid-ups
12 driver.parameter.offdelay: 30
13 driver.parameter.ondelay: 40
14  ...
15 ups.status: OL CHRG
```

Let's ask, using Bash syntax, for a list of the drivers used by upsd:

```
16 rprice@maria:~> for u in $(upsc -l)
17 > do upsc $u driver.name
18 > done
19 usbhid-ups
20 dummy-ups
```

Man page `man upsc` provides further examples.

1.5 Configuration file formats

The components of NUT get their configuration from the following configuration files. The simpler configurations do not use all these files.

- `nut.conf` Nut daemons to be started.
- `ups.conf` Declare the UPS's to be managed by upsd.
- `heartbeat.conf` Used only for heartbeat configurations.
- `upsd.conf` Access control to the upsd daemon.
- `upsd.users` Who has access to the upsd daemon.
- `upsmon.conf` upsmon daemon configuration.
- `upssched.conf` Only used for customised and timer-based setups.
- **upssched-cmd** A script used only for customised and timer-based setups.
- **delayed UPS shutdown** Choice of scripts for delayed UPS shutdown.

NUT parses all the configuration files with a common state machine, which means they all have the following characteristics.

First, most of the programs use an uppercase word to declare a configuration directive. This may be something like `MONITOR`, `NOTIFYCMD`, or `ACCESS`. Case matters 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 so.

The keywords are often followed by values. If you need to set a value to something containing spaces, it has to be contained within "quotes" to keep the parser from splitting the line, e.g.

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

Without the quotes, the parser would only see the first word on the line. Let's say you really need to embed a quote within your directive for some reason. You can do that too.

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

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

When you need to put the \ character into your string, you just escape it.

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

The \ can 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:

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

is the comment character. Anything after an unescaped # is ignored, e.g.

```
25 identity = my#1ups
```

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

```
26 identity = my\#1ups
```

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". Remember that the # character in a password must be escaped:

```
27 password = 12=34#56                Incorrect
28 password = 12\=34\#56                Good
29 password = NUT=Awesome             Incorrect
30 password = "NUT=Awesome"              Good
```

1.5.1 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.

1.6 Mailing list: nut-users

The NUT project offers a mailing list to assist the users. The web page for list administration is https://lists.alioth.debian.org/mailman/listinfo/nut-upsuser.

As always in mailing lists, you get better results if you remember Eric Raymond's good advice which you will find in "How To Ask Questions The Smart Way" at http://www.catb.org/esr/faqs/smart-questions.html.

The NUT mailing lists accept HTML formatted e-mails, but it's better to get into the habit of sending only plain text, since you will meet mailing lists that send HTML to `/dev/null`.

If you want to quote configuration files, please remove comments and blank lines. A command such as `grep ^[^#] upsmon.conf` will do the job. To save you some work, there is ready-made script to prepare a report on a NUT configuration. See `nut-report` script available at http://rogerprice.org/NUT/nut-report.

1.7 NUT has an RFC

On August 8th, 2022, the IETF published RFC 9271 Uninterruptible Power Supply (UPS) Management Protocol – Commands and Responses which describes in detail the commands and responses of the NUT protocol. The RFC follows changes in the technical terms used by the NUT Project and listed in figure 5.

Term used up to NUT 2.7.4	**Term used in NUT 2.8.0, RFC 9271 and this document**
ALREADY-LOGGED-IN	ALREADY-ATTACHED
ALREADY-SSL-MODE	TLS-ALREADY-ENABLED
LOGIN	ATTACH
LOGOUT	DETACH
Master	Primary
NETVER	PROTVER
NUMLOGINS	NUMATTACH
Slave	Secondary

Figure 5: NUT terms changed by NUT 2.8.0 and the RFC.

The RFC uses the term "public power supply" where this text refers to "wall power".

Now that we have the basic ideas of NUT, we are ready to look at the first simple configuration.

2 Simple server with no local users

This chapter extends the general ideas of chapter 1 to provide a fully worked example of a simple configuration. This will in turn form the basis of future chapters. In this simple configuration, the attachment daemon and the management daemon run in the same machine.

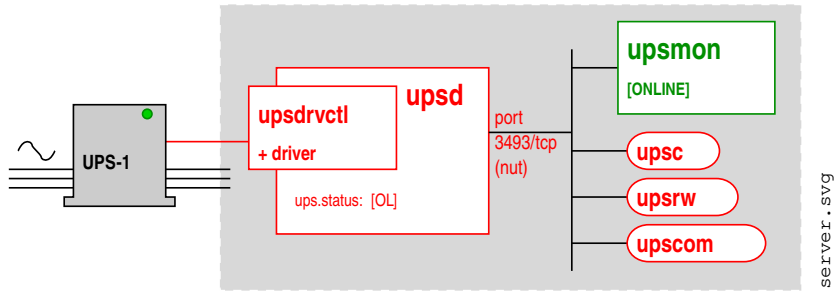


Figure 6: Server with no local users.

Six configuration files specify the operation of NUT in the simple server.

1. The NUT startup configuration: `nut.conf`. Since this file is not strictly a part of NUT, and is common to all configurations, it is discussed separately in appendix A.
2. The upsd UPS declarations: `ups.conf`, see chapter 2.1.
3. The upsd daemon access control; `upsd.conf`, see chapter 2.2.
4. The upsd daemon user declarations: `upsd.users`, see chapter 2.3.
5. The upsmon daemon configuration: `upsmon.conf`, see chapter 2.4.
6. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B.

2.1 Configuration file `ups.conf`, first attempt

```
31 # ups.conf, first attempt
32 [UPS-1]
33    driver = usbhid-ups
34    port = auto
35    desc = "Example Mfg 1600"
```

Figure 7: Configuration file `ups.conf`, first attempt.

This configuration file declares your UPS units. The file described here will do the job, but we will see after we have discussed the shutdown process, that useful improvements are possible.

Line 32 begins a UPS-specific section, and names the UPS unit `UPS-1`. There will as many sections as there are UPS units. Make sure this UPS name matches the name in `upsmon.conf` and in upssched-cmd, which we will meet in later chapters.

Line 33 specifies the driver that upsd will use. For the full list of drivers, see the Hardware Compatibility list and the required drivers at http://www.networkupstools.org/stable-hcl.html.

Line 34 depends on the driver. For the usbhid-ups driver the value is always `auto`. For other drivers, see the man page for that driver.

Line 35 provides a descriptive text for the UPS.

2.2 Configuration file upsd.conf

```
36 # upsd.conf
37 LISTEN 127.0.0.1 3493
38 LISTEN ::1 3493
```

Figure 8: Configuration file upsd.conf.

This configuration file declares on which ports the upsd daemon will listen, and provides a basic access control mechanism.

Line 37 declares that upsd is to listen on it's prefered port for traffic from the localhost. The IP address specifies the interface on which the upsd daemon will listen. The default 127.0.0.1 specifies the loopback interface. It is possible to replace 127.0.0.1 by 0.0.0.0 which says "listen for traffic from all sources" and use your firewall to filter traffic to port 3493. For good security, this file should be accessible to the upsd process only.

If you do not have IPv6, remove or comment out line 38.

2.3 Configuration file upsd.users

```
39 # upsd.users
40 [nut-admin]
41    password = sekret
42    upsmon primary
```

Figure 9: Configuration file upsd.users for a simple server.

This configuration file declares who has write access to the UPS. For good security, ensure that only users nut[4] and root can read and write this file.

Line 40 declares the "user name" of the system administrator who has write access to the UPS's managed by upsd. It is independent of `/etc/passwd`. The upsmon client daemon will use this name to poll and command the UPS's. There may be several names with different levels of access. For this example we only need one.

Line 41 provides the password. You may prefer something better than "sekret". **Warning:** Avoid placing spaces `U+0020` and quotation marks `"` `U+0022` in passwords.

Line 42 declares that this user is the upsmon daemon, and the required set of actions will be set automatically. In this simple configuration daemon upsmon is a `primary`[5] and has authority to shutdown the server. The alternative, "`upsmon secondary`", allows monitoring only, with no shutdown authority.

The configuration file for upsmon must match these declarations for upsmon to operate correctly.

For lots of details, see `man upsd.users`.

[4] This is for Debian 11. See table 104 in appendix C for other user names.

[5] Up to NUT 2.7.4 the primary was known as the "master". The secondary was known as the "slave".

2.4 Configuration file upsmon.conf for a simple server

This configuration file declares how upsmon is to handle NOTIFY events. For good security, ensure that only users nut[6] and root can read and write this file.

```
43 # upsmon.conf
44 MONITOR UPS-1@localhost 1 nut-admin sekret primary
```

Figure 10: Configuration file upsmon.conf for a simple server, part 1 of 5.

On line 44

- The UPS name UPS-1 must correspond to that declared in ups.conf line 32.
- The "power value" 1 is the number of power supplies that this UPS feeds on this system.
- nut-admin is the "user" declared in upsd.users line 40.
- sekret is the password declared in upsd.users line 41.
- primary means this system will shutdown last, allowing any secondaries time to shutdown first. Secondary systems will be discussed in chapter 5. There are no secondaries in this simple configuration.

```
45 SHUTDOWNCMD "/sbin/shutdown -h +0"
46 POWERDOWNFLAG /etc/killpower
```

Figure 11: Configuration file upsmon.conf for a simple server, part 2 of 5.

Line 45 declares the command that is to be used to shut down the server. A second instance of the upsmon daemon running as root will execute this command. Multiple commands are possible, for example `SHUTDOWNCMD "logger -t upsmon.conf \"SHUTDOWNCMD calling /sbin/shutdown to shut down system\" ; /sbin/shutdown -h +0"` will also log the action of SHUTDOWNCMD. Note that internal " have to be escaped.

Line 46 declares a file created by upsmon when running in primary mode when the UPS needs to be powered off. It will be used in more complex configurations. See `man upsmon.conf` for details.

Lines 47-56 assign a text message to each NOTIFY event. Within each message, the marker %s is replaced by the name of the UPS which has produced this event. upsmon passes this message to program wall to notify the system administrator of the event. You can change the default messages to something else if you like. The format is NOTIFYMSG event "*message*" where %s is replaced with the identifier of the UPS in question.

Lines 57-66 declare what is to be done at each NOTIFY event. The declarations, known as "flags" are shown in table 14. You may specify one, two or three flags for each event, in the form FLAG[+FLAG]*, however IGNORE must always be alone.

Note that if you have multiple UPS's, the same actions are to be performed for a given NOTIFY event for all the UPS's. *We will see later that this is not good news.*

[6] This is for Debian 11. See table 104 in appendix C for other user names.

```
47 NOTIFYMSG ONLINE   "UPS %s: On line power."
48 NOTIFYMSG ONBATT   "UPS %s: On battery."
49 NOTIFYMSG LOWBATT  "UPS %s: Battery is low."
50 NOTIFYMSG REPLBATT "UPS %s: Battery needs to be replaced."
51 NOTIFYMSG FSD      "UPS %s: Forced shutdown in progress."
52 NOTIFYMSG SHUTDOWN "Auto logout and shutdown proceeding."
53 NOTIFYMSG COMMOK   "UPS %s: Communications (re-)established."
54 NOTIFYMSG COMMBAD  "UPS %s: Communications lost."
55 NOTIFYMSG NOCOMM   "UPS %s: Not available."
56 NOTIFYMSG NOPARENT "upsmon parent dead, shutdown impossible."
```

Figure 12: Configuration file upsmon.conf for a simple server, part 3 of 5.

```
57 NOTIFYFLAG ONLINE   SYSLOG+WALL
58 NOTIFYFLAG ONBATT   SYSLOG+WALL
59 NOTIFYFLAG LOWBATT  SYSLOG+WALL
60 NOTIFYFLAG REPLBATT SYSLOG+WALL
61 NOTIFYFLAG FSD      SYSLOG+WALL
62 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
63 NOTIFYFLAG COMMOK   SYSLOG+WALL
64 NOTIFYFLAG COMMBAD  SYSLOG+WALL
65 NOTIFYFLAG NOCOMM   SYSLOG+WALL
66 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 13: Configuration file upsmon.conf for a simple server, part 4 of 5.

IGNORE	Don't do anything. Must be the only flag on the line.
SYSLOG	Write the message in the system log.
WALL	Use program wall to send message to terminal users. Note that wall does not support accented letters or non-latin characters.
EXEC	*(Not used for this simple server example).*

Figure 14: Flags declaring what upsmon is to do for NOTIFY events.

```
67 RBWARNTIME 43200
68 NOCOMMWARNTIME 300
69 FINALDELAY 5
```

Figure 15: Configuration file upsmon.conf for a simple server, part 5 of 5.

When a UPS says that it needs to have its battery replaced, upsmon will generate a [REPLBATT] NOTIFY event. Line 67 say that this happens every RBWARNTIME = 43200 seconds (12 hours).

Line 68: Daemon upsmon will trigger a [NOCOMM] NOTIFY event after `NOCOMMWARNTIME` seconds if it can't reach any of the UPS entries in configuration file `upsmon.conf`. It keeps warning you until the situation is fixed.

Line 69: When running in primary mode, upsmon waits this long after sending the [SHUTDOWN] NOTIFY event to warn the users. After the timer elapses, it then runs your `SHUTDOWNCMD` as specified on line 45. If you need to let your users do something in between those events, increase this number. Remember, at this point your UPS battery is almost depleted, so don't make this too big. Alternatively, you can set this very low so you don't wait around when it's time to shut down. Some UPS's don't give much warning for low battery and will require a value of 0 here for a safe shutdown.

For lots and lots of details, see `man upsmon.conf`. See also the file `config-notes.txt` in the distribution.



2.5 The delayed UPS shutdown

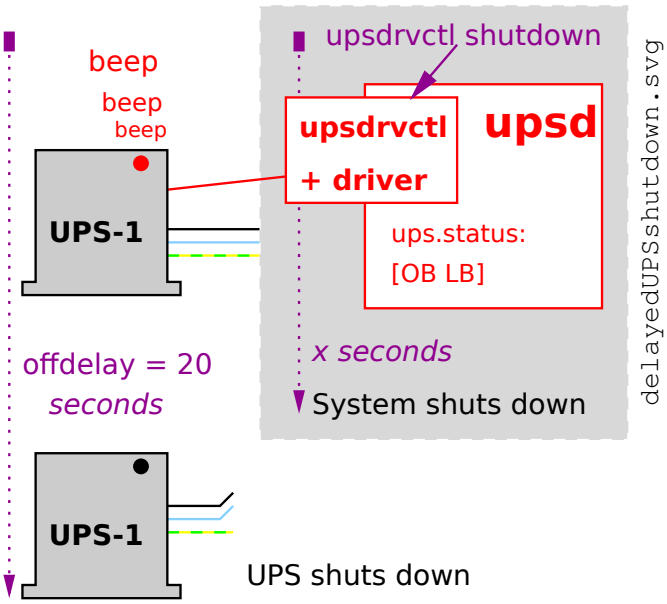


Figure 16: Delayed UPS shutdown.

Somewhere in your distribution, as part of the system shutdown process, there needs to be an action to send a message to the UPS to tell it that some time later, it too will shut down. Note that the UPS does not shutdown at the same time as the system it protects. The UPS shutdown is **delayed**. By default the delay is 20 seconds. We will see in a later chapter how to change this. (Line 77 if you're curious.)

The delayed UPS shutdown command may be from a shell script or a systemd service unit but in all cases the key element is the command `upsdrvctl shutdown`.

Figure 17 shows the shell script supplied by NUT to be placed in a systemd "drop-in" directory for scripts which should be executed as late as possible during a system shutdown. This script is used by openSUSE and Debian 11 and is placed in file `/usr/lib/systemd/system-shutdown/nutshutdown` . systemd detects automatically that a script in one of these "drop-in" directories needs to be executed. There is no need to enable the script.

```
70 #!/bin/sh
71 /usr/sbin/upsmon -K >/dev/null 2>&1 && /usr/sbin/upsdrvctl shutdown
```

Figure 17: openSUSE script for delayed UPS shutdown.

Gentoo users: see Denny Page's post at https://alioth-lists.debian.net/pipermail/nut-upsuser/2018-July/011172.html .

In all these cases, the file name "`nutshutdown`" seems to me to be a misnomer, since it is not NUT which is being shut down, but such naming sloppiness is common.

Warning: This script is executed late in the system shutdown process, and there is no trace in the system log of it's action. If, like the editor, you believe that shutting off power to a system is a major event, and should be logged, then you are invited to replace the script provided by NUT with a systemd service unit as shown in appendix B which will log the delayed shutdown command.

2.6 The shutdown story for a simple server

We are now ready to tell the detailed story of how the server gets shut down when wall power fails, and how it restarts when wall power returns.

1. **Wall power on** The system runs normally. In the attachment daemon the upsd status is [OL]. No NOTIFY event.

   Days, weeks, months go by...

2. **Wall power fails** The server remains operational running on the UPS battery. upsd polls the UPS, and detects status change [OL]→[OB].

3. In the management daemon, upsmon polls upsd, receives [OB] and issues NOTIFY event [ONBATT]. As instructed by line 58, an [ONBATT] message goes to syslog and to program `wall`. The server is still operational running on the UPS battery.

   Minutes go by...

4. **Battery discharges below `battery.charge.low`** The server remains operational, but the UPS battery will not last much longer. upsd polls the UPS, and detects status change [OB]→[OB LB].

5. upsmon polls upsd, receives [OB LB] and issues new NOTIFY event [LOWBATT]. As instructed by line 59 upsmon sends a [LOWBATT] message to syslog and to program `wall`.

6. upsmon decides to command a system shutdown and generates NOTIFY event [SHUTDOWN]. It also sends command `FSD` to upsd to tell any secondaries that they must shut down. There are no secondaries in this simple configuration.

7. upsmon uses command `NUMATTACH`[7] to query the number of secondaries currently operational. In this simple configuration, the reply is 1, there are no secondaries and the primary may proceed to shut down.

8. upsmon waits `FINALDELAY` seconds as specified on line 69.

9. upsmon creates `POWERDOWN` flag specified on line 46.

10. upsmon calls the `SHUTDOWNCMD` specified on line 45.

11. We now enter the scenario described in figure 16. The operating system's shutdown process takes over. During the system shutdown, the Bash script shown in figure 17 or equivalent systemd service unit or some other equivalent runs the command `upsdrvctl shutdown` . This tells the UPS that it is to shut down 20 seconds later.

12. The system powers down, hopefully before the 20 seconds have passed.

13. **UPS shuts down** 20 seconds have passed. With some UPS units, there is an audible "clunk". The UPS outlets are no longer powered. The absence of AC power to the protected

[7] `NUMATTACH` was known as `NUMLOGINS` up to NUT 2.7.4.

system for a sufficient time has the effect of resetting the BIOS options, and in particular the option "Restore power on AC return". This BIOS option will be needed to restart the box. How long is a sufficient time for the BIOS to reset? This depends very much on the box. Some need more than 10 seconds. What if wall power returns before the "sufficient time" has elapsed? The UPS unit will wait until the time specified by the `ondelay` option in file `ups.conf`. This timer, like the `offdelay` timer, starts from the moment the UPS receives the `upsdrvctl shutdown` command. See line 78 in figure 18.

Minutes, hours, days go by...

14. **Wall power returns** Some time later, maybe much later, wall power returns. The UPS reconnects it's outlets to send power to the protected system.
15. The system BIOS option "Restore power on AC return" has hopefully been selected and the system powers up. The bootstrap process of the operating system begins.
16. The operating system starts the NUT daemons upsd and upsmon. Daemon upsd starts the driver(s) and scans the UPS. The UPS status becomes [OL LB]. Daemon upsmon sends command `ATTACH`[8] to upsd to advise upsd that the primary is operational, i.e. the number of attached systems is ≥ 1.
17. After some time, the battery charges above the `battery.charge.low` threshold and upsd declares the status change [OL LB]→[OL]. We are now back in the same situation as state 1 above.



As we saw in figure 16, there is a danger that the system will take longer than 20 seconds to shut down. If that were to happen, the UPS shutdown would provoke a brutal system crash. To alleviate this problem, the next chapter proposes an improved configuration file `ups.conf`.

[8] `ATTACH` was known as `LOGIN` up to NUT 2.7.4.

2.7 Configuration file ups.conf for a simple server, improved

Let's revisit this configuration file which declares your UPS units.

```
72 # ups.conf, improved
73 [UPS-1]
74    driver = usbhid-ups
75    port = auto
76    desc = "Example Mfg 1600"
77    offdelay = 60
78    ondelay = 70
79    lowbatt = 33
```

Figure 18: Configuration file ups.conf, improved.

New line 77 increases from the default 20 secs to 60 secs the time that passes between the **upsdrvctl shutdown** command and the moment the UPS shuts itself down.

Line 78 increases the time that must pass between the **upsdrvctl shutdown** command and the moment when the UPS will react to the return of wall power and turn on the power to the system. Even if wall power returns earlier, the UPS will wait **ondelay = 70** seconds before powering itself on. The default is 30 seconds.

The **ondelay** **must** be greater than the **offdelay**. See **man ups.conf** for more news about this configuration file.

Additional line 79 sets the default value for battery.charge.low. Even if you use command upsrw to set a value for battery.charge.low, usbhid-ups and some other drivers[9] will restore the default, so if you want a permanent change you must change the default. See also chapter 2.10.

2.8 The shutdown story with quick power return

What happens if power returns after the system shuts down but before the UPS delayed shutdown? We pick up the story from state 6.

6. upsmon decides to command a system shutdown and generates NOTIFY event [SHUTDOWN]. It also sends command **FSD** to upsd to tell any secondaries that they must shut down. There are no secondaries in this simple configuration.
7. upsmon uses command **NUMATTACH** to query the number of secondaries currently oerational. In this simple configuration, the reply is 1, since there are no secondaries. The primary may proceed to shut down.
8. upsmon waits **FINALDELAY** seconds as specified on line 69.
9. upsmon creates **POWERDOWN** flag specified on line 46.
10. upsmon calls the **SHUTDOWNCMD** specified on line 45.
11. We now enter the scenario described in figure 16. The operating system's shutdown process takes over. During the system shutdown, the Bash script shown in figure 17 or equivalent systemd service unit or some other equivalent runs the command **upsdrvctl shutdown** . This tells the UPS that it is to shut down **offdelay** seconds later .

[9] List needed

12. The system powers down before `offdelay` seconds have passed.
13. **Wall power returns before the UPS shuts down** Less than `offdelay` seconds have passed. The UPS continues it's shutdown process.
14. After `offdelay` seconds the UPS shuts down, disconnecting it's outlets. The beeping stops. With some UPS units, there is an audible "clunk".

    An interval of `ondelay-offdelay` *seconds later*
15. After `ondelay` seconds the UPS turns itself on, and repowers it's outlets
16. The system BIOS option "restore power on AC return" has hopefully been selected and the system powers up. The bootstrap process of the operating system begins.

The story continues at state 16 in chapter 2.6.

2.9 Utility program upscmd

Utility program upscmd is a command line program for sending commands directly to the UPS. To see what commands your UPS will accept, type `upscmd -l` *`ups-name`* where *`ups-name`* is the name of the UPS as declared in file `ups.conf`, line 32.
For example, to turn on the beeper, use command

```
upscmd -u nut-admin -p sekret UPS-1@localhost beeper.enable
```

where `nut-admin` is the user declared on line 40 and sekret is the l33t password declared on line 41 in file `upsd.users`.

Command upscmd can be dangerous. Make sure that file `upsd.users` can be read and written by root only. See `man upscmd` for more detail.

2.10 Utility program upsrw

Utility program upsrw is a command line program for changing the values of UPS variables. To see which variables may be changed, type `upsrw` *`ups-name`* where *`ups-name`* is the name of the UPS as declared in file `ups.conf`, line 32.
For example, at line 9 we saw that the `battery.charge.low` has been set to 50. We will change this to something less conservative with command

```
upsrw -s battery.charge.low=33 -u nut-admin -p sekret UPS-1@localhost
```

where `nut-admin` is the user declared on line 40 and `sekret` is the password declared on line 41 in file `upsd.users`. Now check that the value has been set with command

```
upsc UPS-1 battery.charge.low
```

which returns the value 33.

Once again, command upsrw can be dangerous. Make sure that file `upsd.users` can be read and written by root only. See `man upsrw` for more detail.

Some drivers, for example usbhid-ups, reset `battery.charge.low` to the default value when they start. To overcome this resistance, add the line `lowbatt = 33` to the UPS definition in file `ups.conf` as shown on line 79.

This chapter has described a basic configuration which is deficient in several ways:

- *NUT messages are only available to those users who are constantly in front of text consoles which display the output of the program `wall`. Systems with users of graphical interfaces which do not display wall output will need stronger techniques.*
- *Program `wall` has not been internationalised. It cannot display letters with accents or any non-latin character.*

Chapter 4 will show how to overcome these difficulties.



3 Server with multiple power supplies

This chapter extends the ideas of chapter 2 to cover a larger server which has multiple, hopefully independent power supplies. The server is capable of running on two or more power supplies, but must be shut down if there are less than two operational. The flexibility of NUT makes this configuration easy: we will describe only the modifications to the configuration in chapter 2.

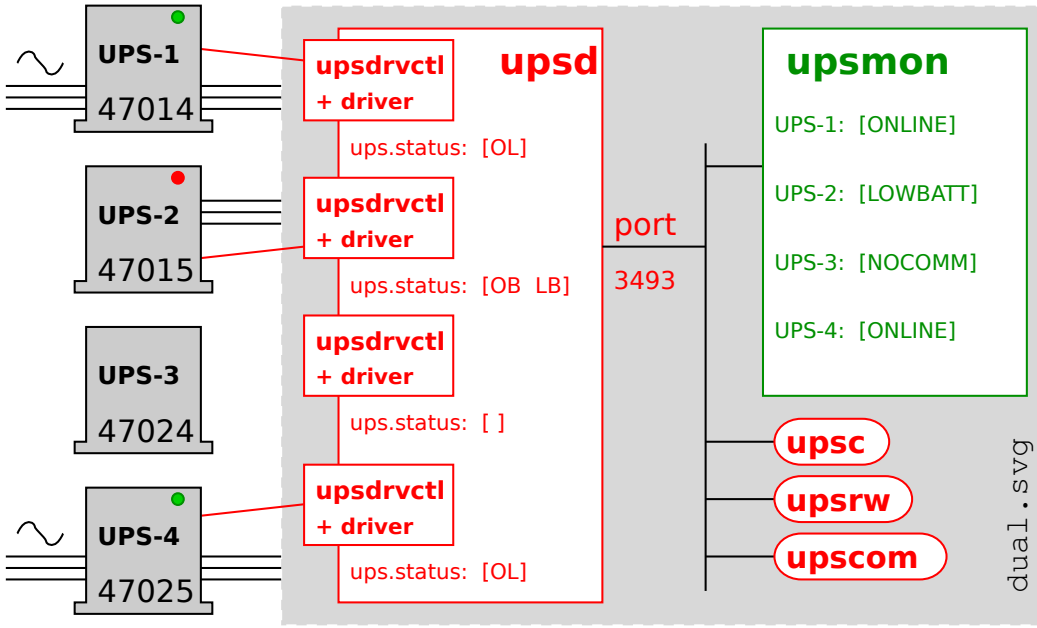


Figure 19: Server with multiple power supplies.

Six configuration files specify the operation of NUT in the server with multiple power supplies.

1. The NUT startup configuration: `nut.conf`. Since this file is not strictly a part of NUT, and is common to all configurations, it is discussed separately in appendix A.
2. The upsd UPS declarations: `ups.conf`, see chapter 3.1.
3. The upsd daemon access control; `upsd.conf` does not change, see chapter 2.2.
4. The upsd daemon user declarations: `upsd.users` do not change, see chapter 2.3.
5. The upsmon daemon configuration: `upsmon.conf`, see chapter 3.2.
6. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B.

3.1 Configuration file `ups.conf` for multiple power supplies

We add additional sections to `ups.conf` to declare the additional UPS units but we need some way of distinguishing them. Assuming the usbhid-ups driver, `man usbhid-ups` describes how this can be done.

```
80  # ups.conf, 4 power supplies
81  [UPS-1]
82     driver = usbhid-ups
83     port = auto
84     desc = "Power supply 1"
85     lowbatt = 33
86     serial = 47014
87  [UPS-2]
88     driver = usbhid-ups
89     port = auto
90     desc = "Power supply 2"
91     lowbatt = 33
92     serial = 47015
```

```
93  [UPS-3]
94     driver = usbhid-ups
95     port = auto
96     desc = "Power supply 3"
97     lowbatt = 33
98     serial = 47024
99  [UPS-4]
100    driver = usbhid-ups
101    port = auto
102    desc = "Power supply 4"
103    lowbatt = 33
104    serial = 47025
```

Figure 20: File ups.conf for multiple power supplies.

Driver usbhid-ups distinguishes multiple UPS units with some combination of the `vendor`, `product`, `serial` and `vendorid` options that it provides. For other drivers, which do not provide the ability to distinguish UPS units, or for UPS units which have no serial number, see the comment by Charles Lepple in NUT issue #597 at https://github.com/networkupstools/nut/issues/597.

Let's assume that the UPS units used in this configuration are sophisticated products and are capable of reporting their serial numbers. You can check this with command `upsc UPS-1 @localhost ups.serial` . In lines 86, 92, 98 and 104 we use this information to distinguish `UPS-1` with `serial = 47014`, `UPS-2` with `serial = 47015`, etc.

See `man ups.conf` and `man usbhid-ups`.

3.2 Configuration file upsmon.conf for multiple power supplies

This configuration file declares how upsmon is to handle NOTIFY events from the UPS units. For good security, ensure that only users nut[10] and root can read and write this file.

```
105 # upsmon.conf, multiple power supplies
106 MONITOR UPS-1@localhost 1 nut-admin sekret primary
107 MONITOR UPS-2@localhost 1 nut-admin sekret primary
108 MONITOR UPS-3@localhost 1 nut-admin sekret primary
109 MONITOR UPS-4@localhost 1 nut-admin sekret primary
110 MINSUPPLIES 2
```

Figure 21: Configuration file upsmon.conf for multiple power supplies, part 1 of 5.

On lines 106-109

[10] This is for Debian 11. See table 104 in appendix C for other user names.

- The UPS names `UPS-1`, `UPS-2`, etc. must correspond to those declared in `ups.conf` lines 81, 87. 93 and 99.
- The "power value" `1` is the number of power supplies that each UPS feeds on this system.
- `nut-admin` is the "user" declared in `upsd.users` line 40.
- `sekret` is the password declared in `upsd.users` line 41.
- `primary` means this system will shutdown last, allowing any secondaries time to shutdown first. Secondary systems will be discussed in chapter 5. There are no secondaries in this configuration.

Line 110, `MINSUPPLIES`, declares that at least two power supplies must be operational, and that if fewer are available, NUT must shut down the server. Figure 19 shows that currently two of the four power supplies are operational. The [OB LB] of `UPS-2`, which would have caused a system shutdown in the case of the simple server in chapter 2 is not sufficient to provoke a system shutdown in this case. `UPS-3` has been disconnected and will be removed in order to paint the wall behind it. (Have you ever worked for Big Business IT, or for Big Government IT?).

The remainder of `upsmon.conf` is the same as that for the simple server of chapter 2, figures 11-15.

3.3 Shutdown conditions for multiple power supplies

```
111 rprice@maria:~> for i in {1..100}
112 > do upsc UPS-1 ups.status 2>&1
113 > sleep 5s
114 > done
115 OL CHRG
116 OL CHRG
              Action: disconnect UPS-1 USB cable
117 Broadcast Message from upsd@maria
118 UPS UPS-1@localhost: Communications lost
119 Error: Data stale
120 Error: Data stale
              Action: reconnect UPS-1 USB cable
121 Broadcast Message from upsd@maria
122 UPS UPS-1@localhost: Communications (re-)established
123 OL CHRG
124 OL CHRG
```

Figure 22: Experiment to show effect of lost UPS. Part 1,

The value of `MINSUPPLIES` is the key element in determining if a server with multiple power supplies should shut down. When all the UPS units can be contacted, and when their `ups.status`

values are known, then it is the count A of those that are active, that is without [LB], which is determinant.

If $A \geq$ `MINSUPPLIES` then OK else shutdown.

`UPS-3`: What is the value of A? The situation for those UPS units such as `UPS-3` is more delicate. If a UPS unit had been reporting the status [OL], then if communication is lost, NUT assumes that the UPS is still operational. Command `upsc UPS-3@localhost ups.status` will return the error message "Error: Data stale", upsmon will raise the NOTIFY event [COMMBAD] and the sysadmin will receive the "Communications lost" message shown on line 54. However this does not count as an [LB].

You can verify this yourself on a simple working configuration such as that of chapter 2 using the Bash command shown on lines 111-114 in figure 22. Disconnecting the USB cable on a healthy UPS does not cause a system shutdown.

```
125 rprice@maria:~> for i in {1..100}
126 > do upsc UPS-1 ups.status 2>&1
127 > sleep 5s
128 > done
129 OL CHRG
130 OL CHRG
              Action: disconnect wall power
131 OB
132 OB
              Action: disconnect UPS-1 USB cable
133 Broadcast Message from upsd@maria
134 UPS UPS-1@localhost: Communications lost
135 Error: Data stale
136 Error: Data stale
              Result: system shutdown
```

Figure 23: Experiment to show effect of lost UPS. Part 2,

However, as shown in figure 23, disconnecting the USB lead on a sick UPS causes a rapid system shutdown. If a UPS unit had been reporting the status [OB], then if communication is lost, NUT assumes that the UPS is about to reach status [OB LB] and calls for a immediate system shutdown.

So the value of A depends not only on the current situation, but also on how the system got into that state.

The moral of our story is that NUT will play safe, but you must be very careful who has access to your server room. We will see in later chapters that there are ways of reinforcing the feedback to the sysadmin.

This chapter has described a complex UPS configuration in isolation, but in practice such a configuration would be just a part of a complete server room, and the use of NUT would have to be integrated with the rest of the server room power management. The layered design of NUT makes this integration possible.



A recent book[11] *for managers on disaster recovery discusses UPS units. On page 559 it says "We chose to have just one UPS do the paging ... We do it on low battery for one of the UPSes that has a 15-minute run-time." Clearly they wanted a timed action, but the only way they could get it was by running down a UPS until it reached* [LB]. *NUT is capable of doing a lot better, as we will show in later chapters.*

[11] "The Backup Book: Disaster Recovery from Desktop to Data Center" by Dorian J. Cougias, E. L. Heiberger, Karsten Koop, Schaser-Vartan Books, 2003, ISBN 0-9729039-0-9, 755 pages.

4 Workstation with local users

This chapter extends the ideas of chapter 2 to provide a fully worked example of a configuration which includes a simple user provided script. This will in turn form the basis for future chapters.

There are two approaches possible for supporting user scripts:

1. *Directly from upsmon using* `NOTIFYCMD`.
2. *Indirectly via upssched and* `CMDSCRIPT`.

We choose the latter since this introduces upssched, which will be needed later.

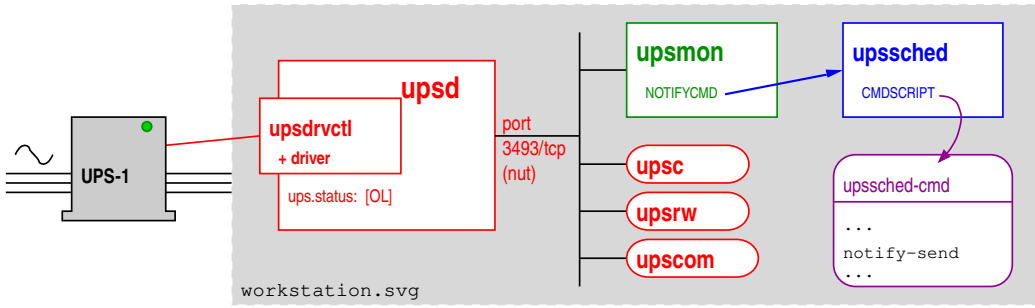


Figure 24: Workstation with local users.

Eight configuration files specify the operation of NUT in the workstation.

1. The NUT startup configuration: `nut.conf`. Since this file is not strictly a part of NUT, and is common to all configurations, it is discussed separately in appendix A.
2. The upsd UPS declarations: The improved file `ups.conf` as given in chapter 2.7 does not change.
3. The upsd daemon access control: File `upsd.conf` as given in chapter 2.2 does not change.
4. The upsd user declarations: File `upsd.users` as given in chapter 2.3 does not change.
5. The upsmon daemon configuration: `upsmon.conf`. See chapter 4.1.
6. The upssched configuration: `upssched.conf`. See chapter 4.2.
7. The upssched-cmd script: see chapter 4.3.
8. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B.

4.1 Configuration file upsmon.conf for a workstation

```
137 # upsmon.conf
138 MONITOR UPS-1@localhost 1 nut-admin sekret primary
139 MINSUPPLIES 1
```

Figure 25: Configuration file upsmon.conf for a workstation, part 1 of 5.

This configuration file declares how upsmon is to handle NOTIFY events. For good security, ensure that only users nut[12] and root can read and write this file.

Line 138 is the same as line 44 in the previous chapter.

On line 139, MINSUPPLIES sets the number of power supplies that must be receiving power to keep this system running. Computers commonly have just one power supply, so the default value of 1 is acceptable. See man upsmon.conf for more details.

```
140 SHUTDOWNCMD "/sbin/shutdown -h +0"
141 NOTIFYCMD /usr/sbin/upssched
142 POLLFREQ 5
143 POLLFREQALERT 5
144 HOSTSYNC 15
145 DEADTIME 15
146 POWERDOWNFLAG /etc/killpower
```

Figure 26: Configuration file upsmon.conf for a workstation, part 2 of 5.

Line 140, identical to line 45 declares the command to be used to shut down the server.

Line 141 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC. The directory/file /usr/sbin/upssched is for Debian 11. Sysadmins for other distributions should check the directory used.

Line 142, POLLFREQ, declares that the upsmon daemon will poll upsd every 5 seconds.

Line 143, POLLFREQALERT, declares that the upsmon daemon will poll upsd every 5 seconds while the UPS in on battery.

Line 144, HOSTSYNC will be used in primary-secondary[13] cooperation, to be discussed in chapter 5.5. The default value is 15 seconds.

Line 145 specifies how long upsmon will allow a UPS to go missing before declaring it "dead". The default is 15 seconds.

Daemon upsmon requires a UPS to provide status information every few seconds as defined by POLLFREQ and POLLFREQALERT. If the status fetch fails, the UPS is marked stale. If it stays stale for more than DEADTIME seconds, the UPS is marked dead.

A dead UPS that was last known to be on battery [OB] is assumed to have changed to a low battery condition [OB]→[OB LB]. This may force a shutdown. Disruptive, but the alternative is

[12] This is for Debian 11. See table 104 in appendix C for other user names.

[13] A secondary is a second, third, ... PC or workstation sharing the same UPS,

barreling ahead into oblivion and crashing when you run out of power. See chapter 3.3 for more discussion.

```
147 NOTIFYMSG ONLINE   "UPS %s: On line power."
148 NOTIFYMSG ONBATT   "UPS %s: On battery."
149 NOTIFYMSG LOWBATT  "UPS %s: Battery is low."
150 NOTIFYMSG REPLBATT "UPS %s: Battery needs to be replaced."
151 NOTIFYMSG FSD      "UPS %s: Forced shutdown in progress."
152 NOTIFYMSG SHUTDOWN "Auto logout and shutdown proceeding."
153 NOTIFYMSG COMMOK   "UPS %s: Communications (re-)established."
154 NOTIFYMSG COMMBAD  "UPS %s: Communications lost."
155 NOTIFYMSG NOCOMM   "UPS %s: Not available."
156 NOTIFYMSG NOPARENT "upsmon parent dead, shutdown impossible."
```

Figure 27: Configuration file upsmon.conf for a workstation, part 3 of 5.

The message texts on lines 147-156 in figure 27 do not change.

```
157 NOTIFYFLAG ONLINE   SYSLOG+WALL+EXEC
158 NOTIFYFLAG ONBATT   SYSLOG+WALL+EXEC
159 NOTIFYFLAG LOWBATT  SYSLOG+WALL+EXEC
160 NOTIFYFLAG REPLBATT SYSLOG+WALL
161 NOTIFYFLAG FSD      SYSLOG+WALL
162 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
163 NOTIFYFLAG COMMOK   SYSLOG+WALL
164 NOTIFYFLAG COMMBAD  SYSLOG+WALL
165 NOTIFYFLAG NOCOMM   SYSLOG+WALL
166 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 28: Configuration file upsmon.conf for a workstation, part 4 of 5.

Lines 157-159 now carry the `EXEC` flag: this flag means that when the NOTIFY event occurs, upsmon calls the program identified by the `NOTIFYCMD` on line 141.

Lines 160-166 do not change.

```
167 RBWARNTIME 43200
168 NOCOMMWARNTIME 300
169 FINALDELAY 5
```

Figure 29: Configuration file upsmon.conf for a workstation, part 5 of 5.

Lines 167-169 are the same as lines 67-69.

4.2 Configuration file upssched.conf for a workstation

The NOTIFY events detected by upsmon and flagged as EXEC in upsmon.conf become events for upssched when NOTIFYCMD points to upssched. The program upssched provides a richer set of actions than upsmon.

The configuration file upssched.conf described here shows only a simple subset of what can be done. We will see more later.

```
170 # upssched.conf
171 CMDSCRIPT /usr/bin/upssched-cmd
172 # PIPEFN LOCKFN suitable for Debian 11
173 PIPEFN /run/nut/upssched.pipe
174 LOCKFN /run/nut/upssched.lock
```

Figure 30: Configuration file upssched.conf for a Debian 11 workstation, Part 1.

On line 171 CMDSCRIPT points to a user script to be called for designated NOTIFY events. This script will receive as argument a user chosen value. The specification /usr/bin/upssched-cmd is for Debian 11. Sysadmins of other distributions should check the directory used.

Line 173 defines PIPEFN which is the file name of a socket used for communication between upsmon and upssched. It is important that the directory be accessible to NUT software and nothing else. I recommend that you use the same directory as is used for communication between upsd and the drivers. Search for the directory which contains the file upsd.pid. You should see at least one socket. See for example the footnote to section 1.3.1.

The value shown on line 173 is for the Debian 11 distribution which places upsd.pid in directory /run/nut/ . As always, sysadmins for other distributions should check the directory used. You should see an additional entry in the directory:

```
175 srw-rw----  1 nut  nut    0 Aug  7 15:57 upssched.pipe=
```

On line 174 the LOCKFN declaration is needed by daemon upsmon to avoid race conditions. The directory should be the same as PIPEFN.

4.2.1 The AT declaration.

```
176 AT ONLINE  UPS-1@localhost EXECUTE online
177 AT ONBATT  UPS-1@localhost EXECUTE onbatt
178 AT LOWBATT UPS-1@localhost EXECUTE lowbatt
```

Figure 31: Configuration file upssched.conf for a Debian 11 workstation, Part 2.

Line 176 introduces the very useful AT declaration provided by upssched.conf. This has the form

`AT` *notifytype UPS-name command*

where

- *notifytype* is a symbol representing a NOTIFY event.
- *UPS-name* can be the special value "`*`" to apply this handler to every possible value of *UPS-name*. I strongly recommend that you do not use this wildcard, since in later chapters we need distinct actions for distinct UPS's.
- The *command* in this case is `EXECUTE`. In later chapters we will see other very useful commands.

Line 176 says what is to be done by upssched for event [ONLINE]. The field "`UPS-1@localhost`" says that it applies to the UPS we are using, and the `EXECUTE` says that the user script specified by `CMDSCRIPT` is to be called with argument "`online`".

Lines 177 and 178 make similar declarations for NOTIFY events [ONBATT] and [LOWBATT].

4.3 Configuration script upssched-cmd for a workstation

When upssched was added to the NUT project, the user defined script was called "upssched-cmd". This is not the most elegant of names but if you use it, people in the NUT community will know immediately what you mean. Ubuntu sysadmins sometimes use `upssched-script` which is better.

```
179 #!/bin/bash -u
180 # upssched-cmd
181 logger -i -t upssched-cmd Calling upssched-cmd $1
```

```
182 UPS="UPS-1"
183 STATUS=$( upsc $UPS ups.status )
184 CHARGE=$( upsc $UPS battery.charge )
185 CHMSG="[$STATUS]:$CHARGE%"
```

```
186 case $1 in
187    online) MSG="$UPS, $CHMSG - power supply has been restored." ;;
188    onbatt) MSG="$UPS, $CHMSG - power failure - save your work!" ;;
189    lowbatt) MSG="$UPS, $CHMSG - shutdown now!" ;;
190    *) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
191       exit 1 ;;
192 esac
193 logger -i -t upssched-cmd $MSG
194 notify-send-all "$MSG"
```

Figure 32: Configuration script upssched-cmd for a workstation.

Figure 32 shows a simple example of the script. Since NUT runs on a wide range of operating systems and distributions, with different default scripting languages, it is wise to declare as on line 179 which scripting language is used.

Logging all calls to this script helps sysadmins to discover what went wrong after the catastrophic failures which in theory should never occur, but which in practice do. Line 181 logs all calls to this script.

Lines 183-185 prepare a Bash variable `CHMSG` which gives the current UPS status and battery charge. This is to be included in messages, so we get a clearer idea of what is happening.

On line 186 the value of the Bash variable `$1` is one of the `EXECUTE` tags defined on lines 176-178.

Lines 187-189 define, for each possible NOTIFY event that upsmon passes on to upssched, a message to be logged and put in front of users. Accented letters and non latin characters are allowed.

Line 193 logs the upssched action, and line 194 calls program notify-send-all to put the message in front of the users. For details of notify-send-all, see appendix D, "Using notify-send". See also `notify-send --help`. There is no man page.

It is important that script upssched-cmd be accessible to NUT software and nothing else. For example the following restrictive ownership and permissions seen on a Debian 11 site:

```
195 root@titan ~ ls -alF /usr/bin/upssched-cmd
196 -rwxr--r-- 1 nut daemon 8444 Aug  6 18:07 /usr/bin/upssched-cmd*
```



4.4 The shutdown story for a workstation

We are now ready to tell the detailed story of how the workstation gets shut down when wall power fails, and how it restarts when wall power returns.

1. **Wall power on** The system runs normally. upsd status is [OL]. No NOTIFY event.

   Days, weeks, months go by...

2. **Wall power fails** The server remains operational running on the UPS battery. upsd polls the UPS, and detects status change [OL]→[OB].
3. upsmon polls upsd, receives status [OB] and issues NOTIFY event [ONBATT]. As instructed by line 158 an [ONBATT] message goes to syslog, to program `wall` and to upssched. The server is still operational, running on the UPS battery.
4. upssched ignores the message it receives and follows the instruction on line 177 to call the user script upssched-cmd with parameter `onbatt`.
5. User script upssched-cmd sees that `$1 = onbatt` and on line 188 sets Bash variable `$MSG` to `UPS-1, [OB DISCHRG]:99% - power failure - save your work!`
6. On line 193, the message is logged, and on line 194 program notify-send-all notifies the users.

   Minutes go by...

7. **Battery discharges below** `battery.charge.low` The server remains operational, but the UPS battery will not last much longer. upsd polls the UPS, and detects status change [OB]→[OB LB].
8. upsmon polls upsd, receives status [OB LB] and issues new NOTIFY event [LOWBATT]. As instructed by line 159 upsmon sends a [LOWBATT] message to syslog, to program `wall` and to upssched.

   The following upssched actions may not occur if the system shutdown is rapid.

9. upssched ignores the message it receives and follows the instruction on line 178 to call the user script upssched-cmd with parameter `lowbatt`.
10. User script upssched-cmd sees that `$1 = lowbatt` and on line 189 sets Bash variable `$MSG` to `UPS-1, [OB DISCHRG LB]:12% - shutdown now!`
11. On line 193, the message is logged, and on line 194 program notify-send notifies the users.

    The shutdown story now continues as for the simple server in state 6.

---

5 Workstations share a UPS

This chapter discusses a variant of the workstation configuration of chapter 4: multiple workstations on the same UPS unit.

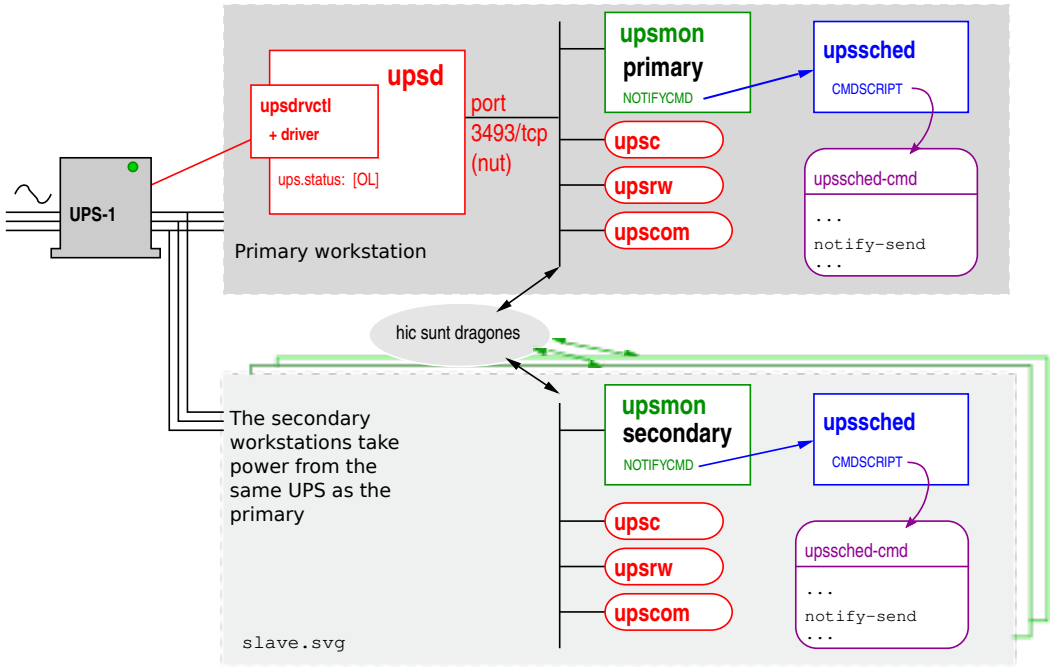


Figure 33: "Secondary" workstations take power from same UPS as "primary".

In this configuration two or more workstations are powered by the same UPS unit. Only one, the "primary", has a control lead to the UPS. The other(s) do not have control leads to the UPS and are known as "secondaries".

Figure 33 shows the arrangement. The NUT configuration for the primary workstation is identical to that of chapter 4.

Five configuration files specify the operation of NUT in the secondary workstation.

1. The NUT startup configuration: `nut.conf`. Since there is no control lead to the UPS, there is no need for upsd or a driver in the secondary. In `nut.conf` declare `MODE=netclient` since only upsmon needs to be started. You will probably need to review your distribution's start-up scripts to achieve this. If upsd is started but without any UPS specified, it usually does no harm. See also appendix A.
2. The upsmon daemon configuration: `upsmon.conf`. See chapter 5.1.
3. The upssched configuration: `upssched.conf`. See chapter 5.2.
4. The upssched-cmd script: see chapter 5.3.
5. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B.

5.1 Configuration file upsmon.conf for a secondary

```
197 # upsmon.conf  -- secondary --
198 MONITOR UPS-1@primary 1 nut-admin sekret secondary
199 MINSUPPLIES 1
```

Figure 34: Configuration file upsmon.conf for a secondary, part 1 of 5.

This configuration file declares how upsmon in the secondary is to handle NOTIFY events coming from the primary. For good security, ensure that only users nut[14] and root can read and write this file.

On line 198

- The UPS name UPS-1 must correspond to that declared in the primary ups.conf, line 32. The fully qualified name *UPS@host* includes the network name of the primary workstation, in this case primary.
- The "power value" 1 is the number of power supplies that this UPS feeds on this system.
- nut-admin is the "user" declared in primary upsd.users line 40.
- sekret is the password declared in primary upsd.users line 41.
- secondary means this system will shutdown first, before the primary.

On line 199, MINSUPPLIES sets the number of power supplies that must be receiving power to keep this system running. Normal computers have just one power supply, so the default value of 1 is acceptable. See chapter 3, and man upsmon.conf in the NUT documentation for more details.

```
200 SHUTDOWNCMD "/sbin/shutdown -h +0"
201 NOTIFYCMD /usr/sbin/upssched
202 POLLFREQ 5
203 POLLFREQALERT 5
204 HOSTSYNC 15
205 DEADTIME 15
206 POWERDOWNFLAG /etc/killpower
```

Figure 35: Configuration file upsmon.conf for a secondary, part 2 of 5.

Line 200, identical to line 45, declares the command to be used to shut down the secondary.

Line 201 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC. Debian administrators would probably specify /sbin/upssched .

Line 202, POLLFREQ, declares that the upsmon daemon will poll upsd in the primary every 5 seconds.

[14] I've seen user upsd rather than nut in distributions. See table 104 in appendix C

Line 203, `POLLFREQALERT`, declares that the upsmon daemon will poll upsd in the primary every 5 seconds while the UPS in on battery.

Line 204, `HOSTSYNC`[15] will be used as a safeguard for the primary-secondary shutdown sequence, preventing the overall shutdown being blocked by an unresponsive secondary, as discussed in chapter 5.4. The default value is 15 seconds.

Line 205 specifies how long the secondary upsmon will allow a UPS to go missing before declaring it "dead". The default is 15 seconds.

Daemon upsmon requires a UPS to provide status information every few seconds as defined by `POLLFREQ` and `POLLFREQALERT`. If the status fetch fails, the UPS is marked stale. If it stays stale for more than `DEADTIME` seconds, the UPS is marked dead.

A dead UPS that was last known to be on battery [OB] is assumed to have changed to a low battery condition [OB]→[OB LB]. This may force a shutdown. Disruptive, but the alternative is barreling ahead into oblivion and crashing when you run out of power. See chapter 3.3 for more discussion.

```
207 NOTIFYMSG ONLINE   "UPS %s: On line power."
208 NOTIFYMSG ONBATT   "UPS %s: On battery."
209 NOTIFYMSG LOWBATT  "UPS %s: Battery is low."
210 NOTIFYMSG REPLBATT "UPS %s: Battery needs to be replaced."
211 NOTIFYMSG FSD      "UPS %s: Forced shutdown in progress."
212 NOTIFYMSG SHUTDOWN "Auto logout and shutdown proceeding."
213 NOTIFYMSG COMMOK   "UPS %s: Communications (re-)established."
214 NOTIFYMSG COMMBAD  "UPS %s: Communications lost."
215 NOTIFYMSG NOCOMM   "UPS %s: Not available."
216 NOTIFYMSG NOPARENT "upsmon parent dead, shutdown impossible."
```

Figure 36: Configuration file upsmon.conf for a secondary, part 3 of 5.

The message texts on lines 207-216 in figure 36 do not change from those in the primary.

```
217 NOTIFYFLAG ONLINE   SYSLOG+WALL+EXEC
218 NOTIFYFLAG ONBATT   SYSLOG+WALL+EXEC
219 NOTIFYFLAG LOWBATT  SYSLOG+WALL+EXEC
220 NOTIFYFLAG REPLBATT SYSLOG+WALL
221 NOTIFYFLAG FSD      SYSLOG+WALL
222 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
223 NOTIFYFLAG COMMOK   SYSLOG+WALL
224 NOTIFYFLAG COMMBAD  SYSLOG+WALL
225 NOTIFYFLAG NOCOMM   SYSLOG+WALL
226 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 37: Configuration file upsmon.conf for a secondary, part 4 of 5.

[15] The name "`HOSTSYNC`" is misleading. It would be clearer if the name were say "`MAXWAITSCNDRY`".

Lines 217-219, which do not change from those in the primary, carry the `EXEC` flag: when the NOTIFY event occurs, secondary upsmon calls the program identified by the `NOTIFYCMD` on line 201.

Lines 220-226 do not change from those in the primary.

```
227 RBWARNTIME 43200
228 NOCOMMWARNTIME 300
229 FINALDELAY 5
```

Figure 38: Configuration file `upsmon.conf` for a secondary, part 5 of 5.

Lines 227-229 are the same as lines 67-69 in the primary.

5.2 Configuration file `upssched.conf` for a secondary

The NOTIFY events detected by secondary upsmon and flagged as `EXEC` in `upsmon.conf` become events for upssched when `NOTIFYCMD` points to upssched. The program upssched provides a richer set of actions than upsmon.

As with the primary in chapter 4, the configuration file `upssched.conf` described here shows only a simple subset of what can be done. We will see more later.

```
230 # upssched.conf  -- secondary  --
231 CMDSCRIPT /usr/sbin/upssched-cmd
232 PIPEFN /run/nut/upssched.pipe
233 LOCKFN /run/nut/upssched.lock
234
235 AT ONLINE  UPS-1@primary EXECUTE online
236 AT ONBATT  UPS-1@primary EXECUTE onbatt
237 AT LOWBATT UPS-1@primary EXECUTE lowbatt
```

Figure 39: Configuration file `upssched.conf` for a secondary.

On line 231, `CMDSCRIPT` points to a user script to be called for designated NOTIFY events. This script will receive as argument a user chosen value.

Line 232 defines `PIPEFN` which is the file name of a socket used for communication between upsmon and upssched. The value shown is for the Debian 11 distribution. For a detailed discussion of `PIPEFN`, see chapter 4.2, line 173.

Daemon upsmon requires the `LOCKFN` declaration on line 233 to avoid race conditions. The directory should be the same as `PIPEFN`.

Line 235 says what upssched should do for NOTIFY event [ONLINE]. The "`UPS-1@primary`" says that it applies to the UPS controlled by the primary, and the `EXECUTE` says that the user script specified by `CMDSCRIPT` is to be called with argument "`online`".

Lines 236 and 237 make similar declarations for NOTIFY events [ONBATT] and [LOWBATT].

5.3 Configuration script upssched-cmd for a secondary

When upssched was added to the NUT project, the user defined script was called "upssched-cmd". This is not the most elegant of names but if you use it, people in the NUT community will know immediately what you mean.

It is important that script upssched-cmd be accessible to NUT software and nothing else.

```
238 #!/bin/bash -u
239 # upssched-cmd  --secondary --
240 logger -i -t upssched-cmd Calling upssched-cmd $1

241 case $1 in
242    online) MSG="UPS-1 - power supply had been restored." ;;
243    onbatt) MSG="UPS-1 - power failure - save your work!" ;;
244    lowbatt) MSG="UPS-1 - shutdown now!" ;;
245    *) logger -i -t upssched-cmd "Bad arg: \"$1\""
246       exit 1 ;;
247 esac
248 logger -i -t upssched-cmd $MSG
249 notify-send-all "$MSG"
```

Figure 40: Configuration script upssched-cmd for a secondary.

Since NUT runs on a wide rage of operating systems and distributions, with different default scripting languages, it is wise to declare as on line 238 which scripting language is used.

Logging all calls to this script helps sysadmins to discover what went wrong after the catastrophic failures which in theory should never occur, but which in practice sometimes do. Line 240 logs all calls to this script.

On line 241 the value of the Bash variable `$1` is one of the `EXECUTE` tags defined on lines 235-237.

Lines 242-244 define, for each possible NOTIFY event that upsmon passes on to upssched, a message to be logged and put in front of users of the secondary. Accented letters and non latin characters are allowed.

Line 248 logs the upssched action, and line 249 calls program notify-send-all to put the message in front of the secondary users. For details of notify-send-all, see appendix D, "Using notify-send". See also `notify-send --help`. There is no man page.

5.4 NUMATTACH: Counting the protected systems

The daemon upsd maintains a count of the number of systems protected by a UPS unit. When a primary or secondary is brought up, it sends command ATTACH to upsd which increases the count by 1. When a system is shut down, it sends command DETACH to upsd and the count decreases by 1. In the configuration shown in figure 33 on page 32, during normal operation the count is 4. The primary upsmon instance polls the number of protected systems using command NUMATTACH. During a complete shutdown, the primary waits until the NUMATTACH value drops to 1 before shutting down itself.

However there is a safeguard against excessive waiting for a non-responsive secondary provided by the HOSTSYNC value declared on line 204.

5.5 Magic: How does the primary shut down the secondaries?

The primary commands the system shutdowns which may be due to an [LB], a timeout (chapter 7), or a sysadmin command. When there are secondaries to be shutdown as well, then the primary expects them to shut down first. But how do the secondaries know that they are to shut down?

When the primary makes the shutdown decision, it places a status symbol [FSD] in variable ups.status in the abstract image of the UPS maintained by it's upsd. The secondary upsmon daemons poll upsd every POLLFREQ seconds as delared on line 142, and when they see the [FSD] symbol, knowing that they are a secondary, they shut down immediately, sending command DETACH to upsd. The primary waits for the secondaries to react and shutdown by polling the NUMATTACH value. The maximum waiting period is specified by HOSTSYNC[16] on line 144. When NUMATTACH = 1, or after the HOSTSYNC time has elapsed, the primary will shut down, even if there is a secondary which has not yet completed it's shutdown. If you meet this problem, you may have to increase the value of HOSTSYNC.

This HOSTSYNC value is also used to keep secondary systems from getting stuck if the primary fails to respond in time. After a UPS becomes critical, e.g. status [OB LB], the secondary will wait up to HOSTSYNC seconds for the primary to set the [FSD] flag. If that timer expires, the secondary will assume that the primary is broken and will shut down anyway. See also man upsmon.conf.



[16] The name "HOSTSYNC" is misleading. It would be clearer if the name were say "MAXWAITSCNDRY".

6 Workstation with heartbeat

The NUT software runs in the background for weeks, months without difficulty and with no messages going the system administrator. "All is well!", but is it? NUT is a collection of pieces and interconnecting protocols. What if one of these pieces has stopped or the protocol blocked? We need something that will check regularly that all is indeed well. The proposed heartbeat does this job.

This chapter supposes that you already have a working configuration for a workstation.

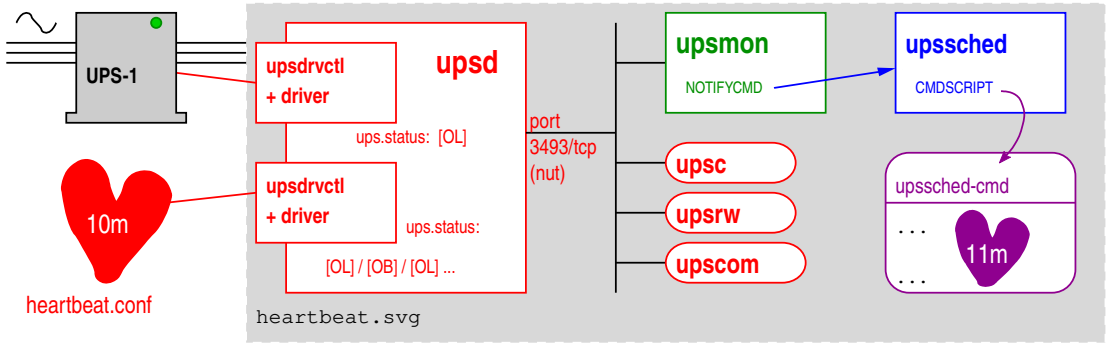


Figure 41: Workstation with heartbeat.

How does it work? NUT program upssched runs permanently as a daemon managing an 11 minute timer. If this timer expires, NUT is broken, and upssched calls user script upssched-cmd which issues wall messages, e-mails, notifications, etc. Meanwhile a dummy (software) UPS is programmed to generate a status change every 10 minutes. This works it's way through the NUT daemons and protocols to reach user script upssched-cmd which then restarts the 11 minute timer. As long as the 10 minute status changes are fully and correctly handled by NUT, the warning message does not go out, but if something breaks, the 11 minute timer elapses.

Nine configuration files specify the operation of NUT in the workstation.

1. The NUT startup configuration: `nut.conf`. See appendix A.
2. The upsd UPS declarations: `ups.conf` will be extended to include the heartbeat. See chapter 6.1.
3. New configuration file `heartbeat.conf` defines the dummy UPS which provides the heartbeat. See chapter 6.2.
4. The upsd daemon access control: File `upsd.conf` as given in chapter 2.2 stays the same.
5. The upsd user declarations: File `upsd.users` as given in chapter 2.3 does not change.
6. The upsmon daemon configuration: `upsmon.conf`. See chapter 6.3.
7. The upssched configuration: `upssched.conf`. See chapter 6.4.
8. The upssched-cmd script: see chapter 6.5.

9. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B.

6.1 Configuration file ups.conf for workstation with heartbeat

We extend this configuration file with an additional section to declare a new UPS unit.

```
250 # ups.conf, heartbeat
251 [UPS-1]
252    driver = usbhid-ups
253    port = auto
254    desc = "Example Mfg Sparkly 1600"
255    offdelay = 60
256    ondelay = 70
257    lowbatt = 33

258 [heartbeat]
259    driver = dummy-ups
260    port = heartbeat.conf
261    mode = dummy-loop
262    desc = "Watch over NUT"
```

Figure 42: Configuration file ups.conf for workstation with heartbeat.

Lines 251-257 are unchanged.

New line 258 declares the new dummy UPS `heartbeat`. This will be a software creation which looks to NUT like a UPS, but which can be programmed with a script, and given arbitrary states.

Line 259 says that this UPS is of type dummy-ups, i.e. a software UPS, for which the scripted behaviour will be in a file specified by the `port` declaration.

Line 260 says that the scripted behaviour is in file `heartbeat.conf` in the same directory as `ups.conf`. Up to 2.7.4 it was traditional in NUT development that such files had file type `.dev`. NUT 2.8.0 has introduced `.seq`, but we are in a production context, not development so we choose a more conventional name.

Starting with 2.8.0, `.dev` no longer implies the looping needed by a heartbeat, this is now called for by `.seq`. The implicit looping behaviour of other file names is no longer defined and must be configured explicitly with a `mode` declaration as shown on line 261.

See `man dummy-ups` for lots of details.

6.2 Configuration file heartbeat.conf for workstation

```
263 # heartbeat.conf -- 10 minute heartbeat
264 ups.status: OL
265 TIMER 300
266 ups.status: OB
267 TIMER 300
```

Figure 43: Configuration file heartbeat.conf for workstation.

Heartbeat definitions are not provided by NUT, you have to create them yourself. Create the new file heartbeat.conf in the same directory as ups.conf. For security, only users nut[17] and root should have write access to this file.

The dummy UPS will cycle continuously through this script.

Lines 264 and 266 flip the ups.status value between [OL] and [OB].

Lines 265 and 267 place a 5 minute time interval between each status change. Remember that $2 \times 300sec = 10min$, the heartbeat period.

6.3 Configuration file upsmon.conf for workstation with heartbeat

The configuration file upsmon.conf is the same as for the workstation in chapter 4, except for an additional MONITOR declaration and a simpler NOTIFYFLAG to avoid flooding the logs.

```
268 # upsmon.conf
269 MONITOR UPS-1@localhost     1 nut-admin sekret primary
270 MONITOR heartbeat@localhost 0 nut-admin sekret primary
271 MINSUPPLIES 1
```

Figure 44: Configuration file upsmon.conf for a workstation with heartbeat.

The change is the addition of line 270 which declares that upsmon is to monitor the heartbeat. Note that the power value is "0" because the heartbeat does not supply power to the workstation.

To avoid flooding your logs, remove the flags SYSLOG and WALL for the [ONLINE] and [ONBATT] NOTIFY events:

```
272 NOTIFYFLAG ONLINE   EXEC
273 NOTIFYFLAG ONBATT   EXEC
```

All the other declarations remain unchanged. This inability of upsmon to provide different behaviours for different UPS's is a weakness, and is why we prefer to make use of upssched which supports precise selection of the UPS in it's AT specification.

[17] Some distributions have been known to use upsd. See table 104 in appendix C for other user names.

6.4 Configuration file upssched.conf for workstation with heartbeat

We use upssched as a daemon to maintain an 11 minute timer which we call `heartbeat-failure-timer`. The timer is kept in memory, and manipulated with the commands `START-TIMER` and `CANCEL-TIMER` which are included in the `AT` facility which NUT provides as a part of `upssched.conf`. See `man upssched.conf`. If this timer completes, upssched calls the user script upssched-cmd with the parameter `heartbeat-failure-timer`, and upssched-cmd will complain that NUT is broken.

The configuration file `upssched.conf` is the same as for the workstation in chapter 4, except for two additional declarations.

```
274 # Restart timer which completes only if the dummy-ups heart beat
275 # has stopped.  See timer values in heartbeat.conf
276 AT ONBATT heartbeat@localhost CANCEL-TIMER heartbeat-failure-timer
277 AT ONBATT heartbeat@localhost START-TIMER  heartbeat-failure-timer 660
```

Figure 45: Configuration file `upssched.conf` for a workstation with heartbeat.

Remember that the very useful `AT` declaration provided by `upssched.conf` has the form

`AT` *notifytype UPS-name command*

On line 276, when upssched receives an [ONBATT] it executes the *command* which is `CANCEL-TIMER heartbeat-failure-timer`. This kills the timer. upssched does not call the user script.

Immediately afterwards, on line 277, and for the same [ONBATT] event, upssched executes the *command* `START-TIMER heartbeat-failure-timer 660` which restarts the `heartbeat-failure-timer` which will run for 660 sec, i.e. 11 minutes. If the timer completes, upssched will call the user script upssched-cmd with parameter `heartbeat-failure-timer`.

Make sure that there are no entries such as

```
278 AT ONLINE * ...
279 AT ONBATT * ...
```

which would be activated by an [ONLINE] or [ONBATT] from the heartbeat UPS. Replace the "*" with the full address of the UPS unit, e.g. `UPS-1@localhost`.

6.5 Script upssched-cmd for workstation with heartbeat

In upssched-cmd, we add additional code to test for completion of the `heartbeat-failure-timer`, and when it completes send a warning to the sysadmin by e-mail, SMS, pigeon, ...

Here is an example of what can be done. Note the e-mail address declarations in the head of the script, and the additional case after "`case $1 in`" beginning on line 297.

On lines 285 and 286, change the e-mail addresses to something that works for you.

Lines 297-304 introduce the `heartbeat-failure-timer` case into the case statement. Line 298 specifies a message to be logged with the current UPS status as defined on lines 288-291.

```
280 #!/bin/bash -u
281 # upssched-cmd for workstation with heartbeat
282 logger -i -t upssched-cmd Calling upssched-cmd $1
283
284 # Send emails to/from these addresses
285 EMAIL_TO="sysadmin@example.com"
286 EMAIL_FROM="upssched-cmd@${HOSTNAME:-nut}.example.com"
287
288 UPS="UPS-1"
289 STATUS=$( upsc $UPS ups.status )
290 CHARGE=$( upsc $UPS battery.charge )
291 CHMSG="[$STATUS]:$CHARGE%"
292
293 case $1 in
294 (online)  MSG="$UPS, $CHMSG - power supply had been restored." ;;
295 (onbatt)  MSG="$UPS, $CHMSG - power failure - save your work!" ;;
296 (lowbatt) MSG="$UPS, $CHMSG - shutdown now!" ;;
297 (heartbeat-failure-timer)
298    MSG="NUT heart beat fails. $CHMSG" ;;
299    # Email to sysadmin
300    MSG1="Hello, upssched-cmd reports NUT heartbeat has failed."
301    MSG2="Current status: $CHMSG \n\n$0 $1"
302    MSG3="\n\n$( ps -elf | grep -E 'ups[dms]|nut' )"
303    echo -e "$MSG1 $MSG2 $MSG3" | /bin/mail -r "$EMAIL_FROM" \
304         -s "NUT heart beat fails. Currently $CHMSG" "$EMAIL_TO"
305 (*) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
306       exit 1 ;;
307 esac
308 logger -i -t upssched-cmd $MSG
309 notify-send-all "$MSG"
```

Figure 46: Configuration script upssched-cmd including heartbeat.

Lines 300-302 compose a message to the sysadmin which is sent on line 303. The message includes the current state of those NUT kernel processes which are operational.

A true sysadmin should not be satisfied with just the heartbeat. "What if the heartbeat dies silently?" We need a further independent check that the normally silent heartbeat is doing it's job.

6.6 For paranoïd sysadmins

We want to check that the heartbeat is in progress. To do so we make use of the permanent presence of a upssched process. Consider the following Bash script:

```
310 #!/bin/bash -u
311 NUT=nut       # openSUSE: "upsd", Debian: "nut"
312 MSGERR="${HOSTNAME:-mybox}: NUT heartbeat fails"
313 MSGOK="${HOSTNAME:-mybox}: NUT heartbeat OK"
314 # Are the heartbeat timers keeping upssched busy?
315 ps -elf | grep "upssched UPS heartbeat" | grep $NUT > /dev/null
316 if [[ $? -ne 0 ]]
317 then wall $MSGERR         # Tell sysadmin the bad news
318      echo -e "$MSGERR" | /bin/mail\
319                          -r heartbeat-watcher@example.com\
320                          -s "$MSGERR" sysadmin@example.com
321      notify-send-all "$MSGERR"
322      sleep 1s
323 else # Tell sysadmin that all is well
324      echo -e "$MSGOK" | /bin/mail\
325                         -r heartbeat-watcher@example.com\
326                         -s "$MSGOK" sysadmin@example.com
327      notify-send-all "$MSGOK"
328 fi
```

Figure 47: Heartbeat watcher.

Line 311 specifies who is the owner of the upssched process. See table 104 for a list of possible owners.

Line 315 will succeed if there is a process managing the heartbeat.

Lines 317, 318 and 321 show three different ways of telling the sysadmin that all is not well with the heartbeat process. Pick which one(s) suit you. See appendix D for a discussion of notify-send-all.

The Bash script requires something like line 329 in `/etc/crontab`:

```
329 1 8 * * *  nut  /usr/local/bin/heartbeat-watcher.sh > /dev/null 2>&1
```

In this example, line 329 declares that the Bash script is to be run at 08:01 hrs every day as user "nut". OpenSUSE might use "upsd". See table 104 for a list of possible users. See also `man crontab(5)`.

This chapter has introduced the timers provided by upssched. We will see in the next chapter that much more can be done with them.

7 Workstation with timed shutdown

All the configurations we have looked at so far have one thing in common. The system shutdown is provoked by UPS status [LB]. *This means that when the system finally shuts down, the battery is depleted. It will still be depleted when wall power returns and the system restarts. This is not a problem if the power supply is inherently reliable, and the power supply will continue long enough to recharge the batteries, but this is not always the case. The maintenance people do not always fix the problem completely on their first visit. In neighbourhoods where lightning strikes frequently, where local industrial activity plays havoc with the voltage, and in neighbourhoods with training schools for backhoe operators, we expect the wall power to fail again, and again.*

In this chapter the criteria for a system shutdown will not be based on the status [LB], *but on the status* [OB] *and an elapsed time.*

It is sometimes said in NUT circles "get the most out of your UPS by hanging on as long as possible". In this chapter we say "get the most out of your UPS by being able to shut down cleanly as often as possible".

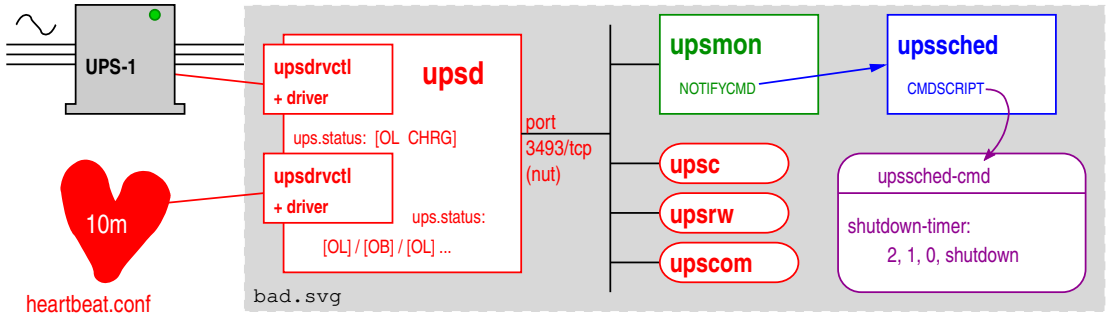


Figure 48: Workstation with timed shutdown.

Nine configuration files specify the operation of NUT in a workstation with timed shutdown. In this chapter we will give these configuration files in full to avoid excessive page turning.

1. The NUT startup configuration: `nut.conf`. Since this file is not strictly a part of NUT, and is common to all configurations, it is discussed separately in appendix A.
2. The upsd UPS declarations `ups.conf`: See chapter 7.1.
3. Configuration file `heartbeat.conf` which defines the dummy UPS providing the heartbeat. See chapter 7.2.
4. The upsd daemon access control `upsd.conf`: See chapter 7.3.
5. The upsd user declarations `upsd.users`: See chapter 7.4.
6. The upsmon daemon configuration: `upsmon.conf`. See chapter 7.5.
7. The upssched configuration: `upssched.conf`. See chapter 7.6.
8. The upssched-cmd script: see chapter 7.7.

9. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B.

7.1 Configuration file ups.conf for workstation with timed shutdown

```
330 # ups.conf, timed shutdown
331 [UPS-1]
332    driver = usbhid-ups
333    port = auto
334    desc = "Bigspark ECO 1600"
335    offdelay = 60
336    ondelay = 70
337    lowbatt = 33
338 [heartbeat]
339    driver = dummy-ups
340    port = heartbeat.conf
341    mode = dummy-loop
342    desc = "Watch over NUT"
```

Figure 49: Configuration file ups.conf for workstation with timed shutdown.

This configuration file includes support for the heartbeat, and is unchanged from that discussed in the previous chapter. See 6.1

Lines 331 and 338 begin a UPS-specific section, and name the UPS unit that upsd will manage. The following lines provides details for each UPS. There will as many sections as there are UPS units. Make sure this name matches the name in upsmon.conf and in upssched-cmd, which we will meet later.

Lines 332 and 339 specify the driver that upsd will use. For the full list of drivers, see the Hardware Compatibility list and the required drivers at http://www.networkupstools.org/stable-hcl.html.

Lines 333 and 340 depend on the driver. For the usbhid-ups driver the value is always `auto`. For the dummy-ups driver, the value is the address of the file which specifies the dummy UPS behaviour. This file should be in the same directory as ups.conf. For other drivers, see the man page for that driver.

Line 341 is needed by NUT 2.8.0 to specify that the program of work described by the file heartbeat.conf is to be repeated endlessly. See `man dummy-ups`.

Lines 334 and 342 provide descriptive texts for the UPS.

For a detailed discussion of `offdelay` and `ondelay` on lines 335-336, see chapter 2.7.

Additional line 337 sets the default value for battery.charge.low. Even if you use command upsrw to set a value for battery.charge.low, usbhid-ups and some other drivers[18] will restore the default, so if you want a permanent change you must change the default. See also chapter 2.10.

[18] List needed

7.2 Configuration file heartbeat.conf for workstation with timed shutdown

Create the new file heartbeat.conf in the same directory as ups.conf.

```
343 # heartbeat.conf
344 # 10 minute heartbeat
345 ups.status: OL
346 TIMER 300
347 ups.status: OB
348 TIMER 300
```

Figure 50: This is the configuration file heartbeat.conf for a workstation with timed shutdown.

This configuration file provides the definition of the heartbeat, and is unchanged from that discussed in chapter 6.2.

The heartbeat definitions are not provided by NUT, you have to create them yourself. Create the new file heartbeat.conf in the same directory as ups.conf. For security, only users nut[19] and root should have write access to this file.

Because it is in mode dummy-loop, the dummy UPS will cycle continuously through this script.

Lines 345 and 347 flip the ups.status value between [OL] and [OB]. Lines 346 and 348 place a 5 minute time interval between each status change. Remember that $2 \times 300sec = 10min$, the heartbeat period.

7.3 Configuration file upsd.conf with timed shutdown

```
349 # upsd.conf
350 LISTEN 127.0.0.1 3493
351 LISTEN ::1 3493
```

Figure 51: Configuration file upsd.conf for workstation with timed shutdown.

This configuration file declares on which ports the upsd daemon will listen, and provides a basic access control mechanism. It does not change from the version shown on lines 37-38.

Line 350 declares that upsd is to listen on it's prefered port for traffic from the localhost. It is possible to replace 127.0.0.1 by 0.0.0.0 which says "listen for traffic from all sources" and use your firewall to filter traffic to port 3493.

If you do not have IPv6, remove or comment out line 351.

[19] This is for Debian 11. See table 104 in appendix C for other user names.

7.4 Configuration file upsd.users with timed shutdown

```
352 # upsd.users
353 [nut-admin]
354     password = sekret
355     upsmon primary
```

Figure 52: This is the configuration file upsd.users for a simple server.

This configuration file declares who has write access to the UPS. It does not change from the version shown in lines 40-42. For good security, ensure that only users nut[20] and root can read and write this file.

Line 353 declares the "user name" of the system administrator who has write access to the UPS's managed by upsd. It is independent of /etc/passwd. The upsmon client daemon will use this name to poll and command the UPS's. There may be several names with different levels of access. For this example we only need one.

Line 354 provides the password. You may prefer something better than "sekret". **Warning:** Avoid placing spaces U+0020 and quotation marks " U+0022 in passwords.

Line 355 declares that this user is the upsmon daemon, and the required set of actions will be set automatically. In this simple configuration daemon upsmon is a primary.

The configuration file for upsmon must match these declaration for upsmon to operate correctly.

For lots of details, see man upsd.users.

7.5 Configuration file upsmon.conf with timed shutdown

The previous chapters have repeatedly modified upsmon.conf so we provide here a complete description of the file, including all previous modifications.

```
356 # upsmon.conf
357 MONITOR UPS-1@localhost     1 nut-admin sekret primary
358 MONITOR heartbeat@localhost 0 nut-admin sekret primary
359 MINSUPPLIES 1
```

Figure 53: Configuration file upsmon.conf with timed shutdown, part 1 of 5.

This configuration file declares how upsmon is to handle NOTIFY events. For good security, ensure that only users nut[21] and root can read and write this file.

On line 357

- The UPS name UPS-1 must correspond to that declared in ups.conf line 331.
- The "power value" 1 is the number of power supplies that this UPS feeds on this system.
- nut-admin is the "user" declared in upsd.users line 353.
- sekret is the password declared in upsd.users line 354.

[20] This is for Debian 11. See table 104 in appendix C for other user names.
[21] This is for Debian 11. See table 104 in appendix C for other user names.

- `primary` means this system will shutdown last, allowing any secondaries time to shutdown first. There are no secondaries in this simple configuration.

Line 358 declares that upsmon is also to monitor the heartbeat.

On line 359, `MINSUPPLIES` sets the number of power supplies that must be receiving power to keep this system running. Normal computers have just one power supply, so the default value of 1 is acceptable. See `man upsmon.conf` and file `big-servers.txt` in the NUT documentation for more details.

```
360 SHUTDOWNCMD "/sbin/shutdown -h +0"
361 NOTIFYCMD /usr/sbin/upssched
362 POLLFREQ 5
363 POLLFREQALERT 5
364 DEADTIME 15
365 POWERDOWNFLAG /etc/killpower
```

Figure 54: Configuration file `upsmon.conf` with timed shutdown, part 2 of 5.

Line 360 declares the command to be used to shut down the server. A second instance of the upsmon daemon running as root will execute this command. Multiple commands are possible, for example `SHUTDOWNCMD "logger -t upsmon.conf \"SHUTDOWNCMD calling /sbin/shutdown to shut down system\" ; /sbin/shutdown -h +0"` will also log the action of `SHUTDOWNCMD`. Note that internal " have to be escaped. Note also that this command will be used in any call to `upsmon -c fsd`. See line 429.

Line 361 says which program is to be invoked when upsmon detects a NOTIFY event flagged as `EXEC`. The example shown is for Debian 11, sysadmins for other distributions should check the directory used.

Line 362, `POLLFREQ`, declares that the upsmon daemon will poll upsd every 5 seconds.

Line 363, `POLLFREQALERT`, declares that the upsmon daemon will poll upsd every 5 seconds while the UPS in on battery.

Line 364, `DEADTIME` specifies how long upsmon will allow a UPS to go missing before declaring it “dead”. The default is 15 seconds.

Daemon upsmon requires a UPS to provide status information every few seconds as defined by `POLLFREQ` and `POLLFREQALERT`. If the status fetch fails, the UPS is marked stale. If it stays stale for more than `DEADTIME` seconds, the UPS is marked dead.

A dead UPS that was last known to be on battery [OB] is assumed to have changed to a low battery condition [OB]→[OB LB]. This may force a shutdown. Disruptive, but the alternative is barreling ahead into oblivion and crashing when you run out of power. See chapter 3.3 for more discussion.

Line 365, `POWERDOWNFLAG` declares a file created by upsmon when running in primary mode when the UPS needs to be powered off. It will be used in more complex configurations. See `man upsmon.conf` for details.

```
366 NOTIFYMSG ONLINE   "UPS %s: On line power."
367 NOTIFYMSG ONBATT   "UPS %s: On battery."
368 NOTIFYMSG LOWBATT  "UPS %s: Battery is low."
369 NOTIFYMSG REPLBATT "UPS %s: Battery needs to be replaced."
370 NOTIFYMSG FSD      "UPS %s: Forced shutdown in progress."
371 NOTIFYMSG SHUTDOWN "Auto logout and shutdown proceeding."
372 NOTIFYMSG COMMOK   "UPS %s: Communications (re-)established."
373 NOTIFYMSG COMMBAD  "UPS %s: Communications lost."
374 NOTIFYMSG NOCOMM   "UPS %s: Not available."
375 NOTIFYMSG NOPARENT "upsmon parent dead, shutdown impossible."
```

Figure 55: Configuration file upsmon.conf with timed shutdown, part 3 of 5.

Lines 366-375 assign a text message to each NOTIFY event. Within each message, the marker %s is replaced by the name of the UPS which has produced this event. upsmon passes this message to program wall to notify the system administrator of the event. You can change the default messages to something else if you like. The format is NOTIFYMSG event "*message*" where %s is replaced with the identifier of the UPS in question. Note that program wall has not been internationalized and does not support accented letters or non latin characters. When the corresponding NOTIFYFLAG contains the symbol EXEC, upsmon also passes the message to the program specified by NOTIFYCMD on line 361.

```
376 NOTIFYFLAG ONLINE   EXEC
377 NOTIFYFLAG ONBATT   EXEC
378 NOTIFYFLAG LOWBATT  SYSLOG+WALL
379 NOTIFYFLAG REPLBATT SYSLOG+WALL
380 NOTIFYFLAG FSD      SYSLOG+WALL
381 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
382 NOTIFYFLAG COMMOK   SYSLOG+WALL
383 NOTIFYFLAG COMMBAD  SYSLOG+WALL
384 NOTIFYFLAG NOCOMM   SYSLOG+WALL
385 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 56: Configuration file upsmon.conf with timed shutdown, part 4 of 5.

Lines 376-385 declare what is to be done at each NOTIFY event. The declarations, known as "flags" are shown in table 14. You may specify one, two or three flags for each event, in the form FLAG[+FLAG]*, however IGNORE must always be alone.

Lines 376-377 carry only the EXEC flag: Since the heartbeat induces a lot of [ONLINE] and [ONBATT] traffic, the SYSLOG option would flood the log and WALL would put far too many useless messages in xterm windows. When the NOTIFY event occurs, EXEC declares that upsmon should call the program identified by the NOTIFYCMD on line 361.

Note that if you have multiple UPS's, the same actions are to be performed for a given NOTIFY event for all the UPS's. *Clearly this is not good news.*

```
386 RBWARNTIME 43200
387 NOCOMMWARNTIME 300
388 FINALDELAY 5
```

Figure 57: Configuration file upsmon.conf with timed shutdown, part 5 of 5.

When a UPS says that it needs to have its battery replaced, upsmon will generate a [REPLBATT] NOTIFY event. Line 386 say that this happens every RBWARNTIME = 43200 seconds (12 hours).

Line 387: Daemon upsmon will trigger a [NOCOMM] NOTIFY event after NOCOMMWARNTIME seconds if it can't reach any of the UPS entries in configuration file upsmon.conf. It keeps warning you until the situation is fixed.

Line 388: When running in primary mode, upsmon waits this long after sending the [SHUTDOWN] NOTIFY event to warn the users. After the timer elapses, it then runs your SHUTDOWNCMD as specified on line 360. If you need to let your users do something in between those events, increase this number. Don't make this too big, even though the battery still has charge. Alternatively, you can set this very low so you don't wait around when it's time to shut down.

For lots and lots of details, see man upsmon.conf. See also the file config-notes.txt in the distribution.

7.6 Configuration file upssched.conf with timed shutdown

The NOTIFY events detected by upsmon and flagged as EXEC in upsmon.conf become events for upssched when NOTIFYCMD points to upssched. The program upssched provides a richer set of actions than upsmon, especially the management of timers.

```
389 # upssched.conf    PIPEFN, LOCKFN for Debian 11
390 CMDSCRIPT /usr/bin/upssched-cmd
391 PIPEFN /run/nut/upssched.pipe
392 LOCKFN /run/nut/upssched.lock
```

Figure 58: Configuration file upssched.conf with timed shutdown, part 1.

On line 390 CMDSCRIPT points to a user script to be called for designated NOTIFY events. The value shown is for Debian 11. Ubuntu sysadmins might see /usr/local/bin/upssched-script. This script will receive as argument a user chosen timer name.

Line 391 defines PIPEFN which is the file name of a socket used for communication between upsmon and upssched. It is important that the directory be accessible to NUT software and nothing else. I recommend that you use the same directory as is used for communication between upsd and the drivers. Search for the directory which contains the file upsd.pid. You should see at least one socket. See for example the footnote to section 1.3.1.

The value shown on line 391 is for the Debian 11 distribution which places upsd.pid in directory /run/nut/ . As always, sysadmins for other distributions should check the directory used. You should see an additional entry in the directory:

```
393 srw-rw----  1 nut  nut    0 Aug  7 15:57 upssched.pipe=
```

Daemon upsmon requires the LOCKFN declaration on line 392 to avoid race conditions. The directory should be the same as PIPEFN.

7.6.1 The AT declaration

```
394 AT ONBATT UPS-1@localhost START-TIMER two-minute-warning-timer 5
395 AT ONBATT UPS-1@localhost START-TIMER one-minute-warning-timer 65
396 AT ONBATT UPS-1@localhost START-TIMER shutdown-timer 125
397
398 AT ONLINE UPS-1@localhost CANCEL-TIMER two-minute-warning-timer
399 AT ONLINE UPS-1@localhost CANCEL-TIMER one-minute-warning-timer
400 AT ONLINE UPS-1@localhost CANCEL-TIMER shutdown-timer
401 AT ONLINE UPS-1@localhost EXECUTE ups-back-on-line
402
403 AT ONBATT heartbeat@localhost CANCEL-TIMER heartbeat-failure-timer
404 AT ONBATT heartbeat@localhost START-TIMER  heartbeat-failure-timer 660
```

Figure 59: Configuration file upssched.conf with timed shutdown, part 2.

Line 394 introduces the very useful AT declaration provided by upssched.conf. This has the form

AT *notifytype UPS-name command*

where

- *notifytype* is a symbol representing a NOTIFY event.
- *UPS-name* can be the special value "*" to apply this handler to every possible value of *UPS-name*. We strongly recommend that you do not use this wildcard, since we need distinct actions for distinct UPS's.
- The *command* values are START-TIMER, CANCEL-TIMER and EXECUTE.

Line 394 says what is to be done by upssched for event [ONBATT]. The field "UPS-1@localhost" says that it applies to the UPS we are using, and the START-TIMER says that upssched is to create and manage a timer called "two-minute-warning-timer" which runs for 5 seconds. When this timer completes, upssched calls the user script specified by CMDSCRIPT with argument "two-minute-warning-timer".

Lines 395 and 396 do the same thing for the 65 second timer one-minute-warning-timer and the 125 second timer shutdown-timer.

Line 398 says what is to be done by upssched for event [ONLINE]. The field "`UPS-1@localhost`" says that it applies to the UPS we are using, and the `CANCEL-TIMER` says that upssched must cancel the timer "`two-minute-warning-timer`". The user script is not called.

Lines 399 and 400 do the same thing for the 65 second timer "`one-minute-warning-timer`" and the 125 second timer "`shutdown-timer`".

Line 401 command `EXECUTE` says that upssched is to call the user script immediately with the argument "`ups-back-on-line`".

On line 403, when upssched receives an [ONBATT] it executes the *command* which is `CANCEL-TIMER heartbeat-failure-timer`. This kills the timer. upssched does not call the user script.

Immediately afterwards, on line 404, and for the same [ONBATT] event, upssched executes the *command* `START-TIMER heartbeat-failure-timer 660` which restarts the `heartbeat-failure-timer` which will run for 660 sec, i.e. 11 minutes. If the timer completes, upssched will call the user script upssched-cmd with parameter `heartbeat-failure-timer`.

7.7 Script upssched-cmd for workstation with timed shutdown

```
405 #!/bin/bash -u
406 # upssched-cmd  Workstation with heartbeat and timed shutdown
407 logger -i -t upssched-cmd Calling upssched-cmd $1

408 # Send emails to/from these addresses
409 EMAIL_TO="sysadmin@example.com"
410 EMAIL_FROM="upssched-cmd@${HOSTNAME:-nut}.example.com"

411 UPS="UPS-1"
412 STATUS=$( upsc $UPS ups.status )
413 CHARGE=$( upsc $UPS battery.charge )
414 CHMSG="[$STATUS]:$CHARGE%"
```

Figure 60: Configuration script upssched-cmd for timed shutdown, 1 of 2.

The user script upssched-cmd, the example is in Bash, manages the completion of the timers `two-minute-warning-timer`, `one-minute-warning-timer`, `shutdown-timer`, `ups-back-on-line` and `heartbeat-failure-timer`. Here is an complete example of what can be done. You will probably need to modify this for your own use. Note that this script could be written in the language of your choice, as long as the resulting program is able to receive the timer names as a parameter, send e-mails and log and notify the users of messages. Bash has the advantage of being widely available and is understood by many sysadmins.

On lines 409 and 410, change the e-mail addresses to something that works for you.

Lines 411-414 prepare a Bash variable `CHMSG` which gives the current UPS status and battery charge. This is to be included in messages, so we get a clearer idea of what is happening.

```
415 case $1 in
416 (heartbeat-failure-timer)
417    MSG="NUT heart beat fails. $CHMSG" ;;
418    MSG1="Hello, upssched-cmd reports NUT heartbeat has failed."
419    MSG2="Current status: $CHMSG \n\n$0 $1"
420    MSG3="\n\n$( ps -elf | grep -E 'ups[dms]|nut' )"
421    echo -e "$MSG1 $MSG2 $MSG3" | /bin/mail -r "$EMAIL_FROM" \
422         -s "NUT heart beat fails. Currently $CHMSG" "$EMAIL_TO" ;;
```

```
423 (two-minute-warning-timer)
424    MSG="Possible shutdown in 2 minutes. Save your work! $CHMSG" ;;
425 (one-minute-warning-timer)
426    MSG="Probable shutdown in 1 minute. Save your work! $CHMSG" ;;
427 (shutdown-timer)
428    MSG="Power  failure shutdown: Calling upsmon -c fsd, $CHMSG"
429    /usr/sbin/upsmon -c fsd ;;
430 (ups-back-on-line)
431    MSG="Power back, shutdown cancelled. $CHMSG"  ;;
432 (*) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
433       exit 1 ;;
434 esac
435 logger -i -t upssched-cmd $MSG
436 notify-send-all "$MSG"
```

Figure 61: Configuration script upssched-cmd for timed shutdown, 2 of 2.

Lines 416-422 introduce the `heartbeat-failure-timer` case into the case statement. Line 417 specifies a message to be logged with the current UPS status as defined on lines 411-414.

Lines 418-420 compose a message to the sysadmin which is sent on line 421. The message includes the current state of those NUT kernel processes which are operational.

7.7.1 The timed shutdown

The cases at lines 423 and 425 specify warnings to be notified to the users when the `two-minute-warning-timer` and `one-minute-warning-timer` complete.

Beginning at line 427 we prepare a message which the user may not see, since we call for an immediate shutdown. The UPS may well be almost fully charged, but the shutdown is now, leaving enough charge for further shutdowns in the near future.

Note on line 429 that we use upsmon to shut down the system. This automatically takes into account any secondary systems which need to be shut down as well. The command `upsmon -c fsd` will call the command specified by the `SHUTDOWNCMD` declaration on line 360.

Line 430 prepares a message that notify-send-all will put in front of the users to tell them to get back to work since wall power has returned. See appendix D for a discussion of notify-send-all.

7.8 The timed shutdown story

We now tell the detailed story of how the workstation gets shut down when wall power fails, and how it restarts when wall power returns.

1. **Wall power on** The system runs normally. upsd status is [OL]. No NOTIFY event.

   Days, weeks, months go by...

2. **Wall power fails** The workstation remains operational running on the UPS battery. upsd polls the UPS, and detects status change [OL]→[OB].
3. upsmon polls upsd, receives status [OB] and issues NOTIFY event [ONBATT]. As instructed by line 377 upsmon calls upssched, specified by `NOTIFYCMD` on line 361. Note that there is no `wall` message and no logging by upsmon.
4. upssched matches the NOTIFY event [ONBATT] and the UPS name `UPS-1@localhost` with the three `AT` specifications on lines 394-396. Three timers start: `two-minute-warning-timer`, `one-minute-warning-timer` and `shutdown-timer`, managed in memory by upssched.

   5 seconds go by...

5. `two-minute-warning-timer` completes, and upssched calls the user script upsschedcmd specified by `CMDSCRIPT` on line 390 with the timer name as argument. In the script, this matches the case on line 423 which defines a suitable warning message in Bash variable `MSG`. Line 435 logs this message and line 436 puts it in front of the users. The workstation continues to operate on battery power.

   60 seconds go by...

6. `one-minute-warning-timer` completes, and upssched calls the user script upsschedcmd with the timer name as argument. In the script, this matches the case on line 425 which defines a stronger warning message in Bash variable `MSG`. Line 435 logs this message and line 436 puts it in front of the users. The workstation continues to operate on battery power.

   60 seconds go by...

7. `shutdown-timer` completes, and upssched calls the user script upsschedcmd with the timer name as argument. In the script, this matches the case on line 427 which defines an ultimate warning message in Bash variable `MSG`, and then calls upsmon for a system shutdown. Line 435 logs message `MSG` and line 436 puts it in front of the users. The workstation continues to operate on battery power during the shutdown. If wall power returns, it is now too late to call off the shutdown procedure.
8. upsmon commands a system shutdown and generates NOTIFY event [SHUTDOWN].
9. upsmon waits `FINALDELAY` seconds as specified on line 388.
10. upsmon creates `POWERDOWN` flag specified on line 365.
11. upsmon calls the `SHUTDOWNCMD` specified on line 360.

12. We now enter the scenario described in figure 16. The operating system's shutdown process takes over. During the system shutdown, the Bash script shown in figure 17 or equivalent systemd service unit or some other equivalent runs the command `upsdrvctl shutdown` . This tells the UPS that it is to shut down `offdelay` seconds later as specified on line 335.
13. The system powers down, hopefully before the `offdelay` seconds have passed.
14. **UPS shuts down** `offdelay` seconds have passed. With some UPS units, there is an audible "clunk". The UPS outlets are no longer powered.

    Minutes, hours, days go by...
15. **Wall power returns** Some time later, maybe much later, wall power returns. The UPS reconnects it's outlets to send power to the protected system.
16. The system BIOS option "restore power on AC return" has hopefully been selected and the system powers up. The bootstrap process of the operating system begins.
17. The operating system starts the NUT daemons upsd and upsmon. Daemon upsd scans the UPS and the status becomes [OL]. We are now back in the same situation as state 1 above.
18. We hope that the battery has retained sufficient charge to complete further timed shutdown cycles, but if it hasn't, then at the next power failure, upsd will detect the status [OB LB], upsmon will receive status [OB LB] and issue a [LOWBATT] and will begin the system shutdown process used by the simple server of chapter 2. This system shutdown will override any upssched timed process.



8 Workstation with additional equipment

The time has come to look at a more ambitious configuration, with multiple UPS's and multiple computer systems. NUT has been designed as an assembly of components each performing a distinct part of the operation. We now see that this design allows NUT to adapt and perform well in complex configurations.

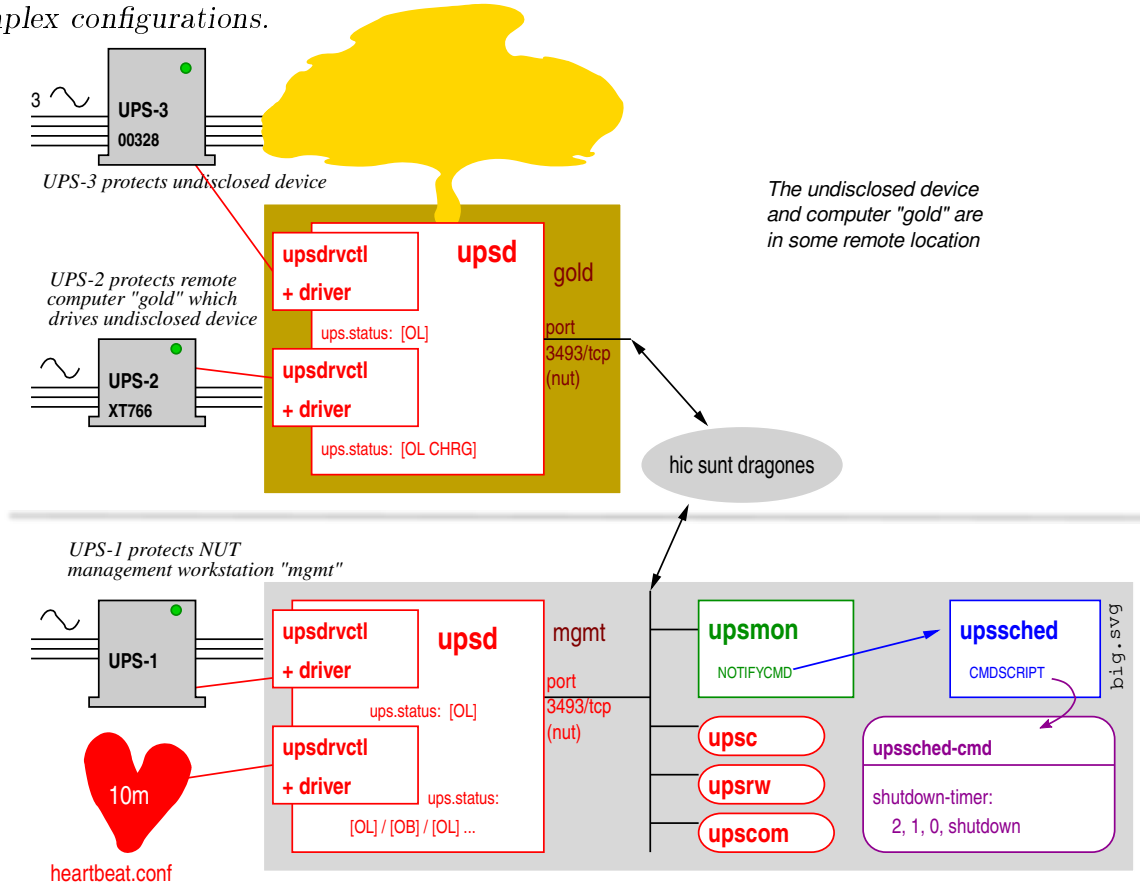


Figure 62: Workstation with additional equipment.

The configuration is for an industrial application in which some undisclosed industrial equipment is protected by a UPS (`UPS-3`), and is also driven by a computer system having it's own UPS (`UPS-2`). This equipment with the driving computer is at a remote site, code name gold. Overall management is from a computer at a different, administrative site. We will call the management system mgmt.

Computer mgmt is represented here as if it were a single machine, but it could well be duplicated at different sites for reliability. Two (or more) mgmt systems may monitor a single gold production machine.

Fourteen configuration files specify the operation of NUT in the production and management machines.

1. gold: The NUT startup configuration: `nut.conf`. This file is not strictly a part of NUT, and is common to all configurations. See chapter 8.1 and appendix A.
2. gold: The upsd UPS declarations `ups.conf`: See chapter 8.2.
3. gold: The upsd daemon access control `upsd.conf`: See chapter 8.3.
4. gold: The upsd user declarations `upsd.users`: See chapter 8.4.
5. gold: The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B. The shutdown script for the undisclosed device is beyond the scope of this text.
6. mgmt: The NUT startup configuration: `nut.conf`. This file is not strictly a part of NUT, and is common to all configurations. See chapter 8.1 also appendix A.
7. mgmt: The upsd UPS declarations `ups.conf`: See chapter 8.2.
8. mgmt: The upsd heartbeat declaration `heartbeat.conf`: See chapter 8.2.
9. mgmt: The upsd daemon access control `upsd.conf`: See chapter 8.3.
10. mgmt: The upsd user declarations `upsd.users`: See chapter 8.4.
11. mgmt: The upsmon daemon configuration `upsmon.conf`: See chapter 8.5.
12. mgmt: The upssched configuration `upssched.conf`: See chapter 8.6.
13. mgmt: The upssched-cmd script: See chapter 8.7.
14. mgmt: The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix B.

8.1 Configuration files `nut.conf`

The first configuration files say which parts of the NUT are to be started.

gold

```
437 # nut.conf  -- gold --
438 MODE=netserver
```

Figure 63: File `nut.conf` for gold.

mgmt

```
439 # nut.conf  -- mgmt --
440 MODE=standalone
```

Figure 64: Files `nut.conf` for mgmt.

Strictly speaking, this file is not for NUT, but for the process which starts NUT. The initialization process is expected to source this file to know which parts of nut are to be started. Some distributions, e.g. openSUSE, ignore this file and start the three NUT layers driver, upsd and upsmon. They assume that `MODE=standalone`.

This is probably satisfactory for mgmt, but for gold you should review line 438 and the init/systemd startup of the NUT software to ensure that only the upsd and driver daemons get started. See appendix A. See also `man nut.conf`.

8.2 Configuration files ups.conf and heartbeat.conf

These configuration files declare which UPS's are to be managed by the instances of NUT.

gold

```
441 # ups.conf  -- gold --
442 [UPS-3]
443    driver = usbhid-ups
444    port = auto
445    desc = "Huge 3 phase"
446    offdelay = 20
447    ondelay = 30
448    lowbatt = 33
449    serial = 00328
450
451 [UPS-2]
452    driver = usbhid-ups
453    port = auto
454    desc = "Small monophase"
455    offdelay = 20
456    ondelay = 30
457    lowbatt = 33
458    serial = XT766
```

Figure 65: File ups.conf for gold.

mgmt

```
459 # ups.conf  -- mgmt --
460 [UPS-1]
461    driver = usbhid-ups
462    port = auto
463    desc = "BigSpark ECO 1600"
464    offdelay = 60
465    ondelay = 70
466    lowbatt = 33
467
468 [heartbeat]
469    driver = dummy-ups
470    port = heartbeat.conf
471    mode = dummy-loop
472    desc = "Watch over NUT"
```

Figure 66: File ups.conf for mgmt.

```
473 # heartbeat.conf -- 10 min
474 ups.status: OL
475 TIMER 300
476 ups.status: OB
477 TIMER 300
```

Figure 67: heartbeat.conf for mgmt.

gold: On lines 442-451 we offer specimen definitions for `UPS-3` and `UPS-2`. You will need to review these to take into account the UPS's you are using. Lines 452 and 443 specify the drivers that upsd will use. For the full list of drivers, see the Hardware Compatibility list and the required drivers at http://www.networkupstools.org/stable-hcl.html.

The `offdelay` and `ondelay` on lines 446-447 and 455-456 are given their default values. You may need something different. See the discussion in chapter 2.5 of the delayed UPS shutdown.

In order to distinguish the two USB attached UPS units on gold, we specify their serial numbers on lines 449 and 458. See `man usbhid-ups`.

mgmt: On lines 460-465 we offer a specimen definition for `UPS-1` and on lines 474-477 we propose the dummy UPS "heartbeat" discussed in chapter 6. The heartbeat requires the definition file heartbeat.conf, lines 474-477, to be placed in the same directory as ups.conf.

8.3 Configuration files upsd.conf

gold

```
478 # upsd.conf   -- gold --
479 LISTEN 10.8.0.5 3493
480 LISTEN X::Y::Z 3493
```

Figure 68: File upsd.conf for gold.

mgmt

```
481 # upsd.conf   -- mgmt --
482 LISTEN 127.0.0.1 3493
483 LISTEN ::1 3493
```

Figure 69: File upsd.conf for mgmt.

This configuration file declares on which ports the upsd daemon will listen, and provides a basic access control mechanism. You will need a secure means of accessing gold from mgmt. This could be for example through an SSH tunnel or over a VPN. The limited access defined by the LISTEN directive is part of a defense in depth.

gold: Line 479 declares that upsd is to listen on a prefered port for traffic from mgmt. The example is for the tun0 interface of an OpenVPN secure network. See https://openvpn.net/ . It is possible to specify 0.0.0.0 which says "listen for traffic from all sources" and use your firewall to filter traffic to port 3493. You must modify lines 479 and 480 for your own needs.

mgmt: Line 482 declares that upsd is to listen on it's prefered port for traffic from the localhost. It is possible to replace 127.0.0.1 by 0.0.0.0 which says "listen for traffic from all sources" and use your firewall to filter traffic to port 3493.

If you do not have IPv6, remove or comment out lines 480 and 483.

See man upsd.conf for more detail, and a description of the OpenSSL support.

8.4 Configuration files upsd.users

gold

```
484 # upsd.users   -- gold --
485 [nut-admin]
486     password = sekret
487     upsmon primary
```

Figure 70: File upsd.users for gold.

mgmt

```
488 # upsd.users   -- mgmt --
489 [nut-admin]
490     password = sekret
491     upsmon primary
```

Figure 71: File upsd.users for mgmt.

This configuration file declares who has write access to the UPS. The "user name" used in these files is independent of /etc/passwd. For good security, ensure that only users nut[22] and root can read and write this file. The configuration files for upsmon must match these declarations for upsmon to operate correctly.

For lots of details, see man upsd.users.

gold: Line 485 declares the "user name" of the system administrator who has write access to UPS-2 and UPS-3 managed by upsd. The upsmon client daemon in mgmt will use this name to poll and command the UPS's.

Line 486 provides the password. You may prefer something better than "sekret". **Warning:** Avoid placing spaces U+0020 and quotation marks " U+0022 in passwords.

[22] This is for Debian 11. See table 104 in appendix C for other user names.

Line 487 declares the type of relationship between the upsd daemon on gold and the upsmon in mgmt which has the authority to shutdown gold. The declaration "`upsmon secondary`" would allow monitoring but not shutdown. See `man upsd.users`. See also `man upsmon` section UPS DEFINITIONS, but our configuration is not exactly what that man page refers to.

mgmt: Line 489 declares the "user name" of the system administrator who has write access to `UPS-1` and to the heartbeat managed by upsd.

Line 490 provides another uberl33t password.

Line 491 declares the type of relationship between the upsd daemon and upsmon which has the authority to shutdown mgmt.

8.5 Configuration file upsmon.conf

The previous chapters have repeatedly modified upsmon.conf so we provide here a complete description of the file.

```
492 # upsmon.conf  -- mgmt --
493 MONITOR UPS-3@gold          0 nut-admin sekret primary
494 MONITOR UPS-2@gold          0 nut-admin sekret primary
495 MONITOR UPS-1@localhost     1 nut-admin sekret primary
496 MONITOR heartbeat@localhost 0 nut-admin sekret primary
497 MINSUPPLIES 1
```

Figure 72: Configuration file upsmon.conf for mgmt, part 1 of 5.

This configuration file declares how upsmon in mgmt is to handle NOTIFY events from gold and from mgmt itself. For good security, ensure that only users nut[23] and root can read and write this file.

Line 493 specifies that upsmon on mgmt will monitor `UPS-3` which supplies power to the undisclosed device.

- The UPS name `UPS-3` must correspond to that declared in ups.conf line 456.
- The "power value" `1` is the number of power supplies that this UPS feeds on the local system. A "power value" of 0 means that the `UPS-3` does not supply power to mgmt.
- `nut-admin` is the "user" declared in upsd.users line 485.
- `sekret` is the l33t password declared in upsd.users line 486.
- `primary` means this system will shutdown last, allowing any secondaries time to shutdown first. There are no secondaries on gold.

Line 494 specifies that upsmon on mgmt will also monitor `UPS-2` which supplies the gold computer.

[23] This is for Debian 11. See table 104 in appendix C for other user names.

Line 495 specifies that upsmon on mgmt will monitor UPS-1 which supplies power to mgmt itself. Note the "power value" of 1.

Line 496 declares that upsmon is also to monitor the heartbeat.

On line 497, MINSUPPLIES sets the number of power supplies that must be receiving power to keep the mgmt system running. A lot of computers have just one power supply, so the default value of 1 is acceptable. See man upsmon.conf and file big-servers.txt in the NUT documentation for more details.

```
498 SHUTDOWNCMD "/sbin/shutdown -h +0"
499 NOTIFYCMD /usr/sbin/upssched
500 POLLFREQ 5
501 POLLFREQALERT 5
502 DEADTIME 15
503 POWERDOWNFLAG /etc/killpower
```

Figure 73: Configuration file upsmon.conf for mgmt, part 2 of 5.

Line 498 declares the command to be used to shut down mgmt. A second instance of the upsmon daemon running as root on mgmt will execute this command. Multiple commands are possible, for example SHUTDOWNCMD "logger -t upsmon.conf \"SHUTDOWNCMD calling /sbin/shutdown to shut down system\" ; /sbin/shutdown -h +0" will also log the action of SHUTDOWNCMD. Note that internal " have to be escaped. Note also that any calls of the command upsmon -c fsd will also execute this command. See line 576.

The shutdown command for gold is not specified in upsmon.conf. It appears in the user script upssched-cmd in chapter 8.7.

Line 499 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC.

Line 500, POLLFREQ, declares that the upsmon daemon will poll upsd in gold and in mgmt every 5 seconds.

Line 501, POLLFREQALERT, declares that the upsmon daemon will poll the upsd daemons every 5 seconds while any UPS in on battery.

Line 502, DEADTIME specifies how long upsmon will allow a UPS to go missing before declaring it "dead". The default is 15 seconds.

Daemon upsmon requires a UPS to provide status information every few seconds as defined by POLLFREQ and POLLFREQALERT. If the status fetch fails, the UPS is marked stale. If it stays stale for more than DEADTIME seconds, the UPS is marked dead.

A dead UPS-1 that was last known to be on battery [OB] is assumed to have changed to a low battery condition [OB]→[OB LB]. This may force a shutdown of mgmt. Disruptive, but the alternative is barreling ahead into oblivion and crashing when you run out of power. See chapter 3.3 for more discussion.

Line 503, POWERDOWNFLAG declares a file created by upsmon when running in primary mode when UPS-1 needs to be powered off. See man upsmon.conf for details.

```
504 NOTIFYMSG ONLINE   "UPS %s: On line power."
505 NOTIFYMSG ONBATT   "UPS %s: On battery."
506 NOTIFYMSG LOWBATT  "UPS %s: Battery is low."
507 NOTIFYMSG REPLBATT "UPS %s: Battery needs to be replaced."
508 NOTIFYMSG FSD      "UPS %s: Forced shutdown in progress."
509 NOTIFYMSG SHUTDOWN "Auto logout and shutdown proceeding."
510 NOTIFYMSG COMMOK   "UPS %s: Communications (re-)established."
511 NOTIFYMSG COMMBAD  "UPS %s: Communications lost."
512 NOTIFYMSG NOCOMM   "UPS %s: Not available."
513 NOTIFYMSG NOPARENT "upsmon parent dead, shutdown impossible."
```

Figure 74: Configuration file upsmon.conf for mgmt, part 3 of 5.

Lines 504-513 assign a text message to each NOTIFY event. Within each message, the marker `%s` is replaced by the name of the UPS which has produced this event. On mgmt upsmon passes this message to program `wall` to notify the system administrator of the event. You can change the default messages to something else if you like. The format is `NOTIFYMSG event "`*`message`*`"` where `%s` is replaced with the identifier of the UPS in question. Note that program `wall` has not been internationalized and does not support accented letters or non latin characters. When the corresponding `NOTIFYFLAG` contains the symbol `EXEC`, upsmon also passes the message to the program specified by `NOTIFYCMD` on line 499.

```
514 NOTIFYFLAG ONLINE   EXEC
515 NOTIFYFLAG ONBATT   EXEC
516 NOTIFYFLAG LOWBATT  SYSLOG+WALL
517 NOTIFYFLAG REPLBATT SYSLOG+WALL
518 NOTIFYFLAG FSD      SYSLOG+WALL
519 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
520 NOTIFYFLAG COMMOK   SYSLOG+WALL
521 NOTIFYFLAG COMMBAD  SYSLOG+WALL
522 NOTIFYFLAG NOCOMM   SYSLOG+WALL
523 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 75: Configuration file upsmon.conf for mgmt, part 4 of 5.

Lines 514-523 declare what is to be done at each NOTIFY event. The declarations, known as "flags" are shown in table 14. You may specify one, two or three flags for each event, in the form `FLAG[+FLAG]*`, however `IGNORE` must always be alone.

Lines 514-515 carry only the `EXEC` flag: Since the heartbeat induces a lot of [ONLINE] and [ONBATT] traffic, the `SYSLOG` option would flood the log and `WALL` would put far too many useless messages in xterm windows. When the NOTIFY event occurs, `EXEC` declares that upsmon should call the program identified by the `NOTIFYCMD` on line 499.

Note that if you have multiple UPS's, the same actions are to be performed for a given NOTIFY event for all the UPS's. *Once again, we see that this is not good news.*

```
524 RBWARNTIME 43200
525 NOCOMMWARNTIME 300
526 FINALDELAY 5
```

Figure 76: Configuration file upsmon.conf for mgmt, part 5 of 5.

When a UPS says that it needs to have its battery replaced, upsmon will generate a [REPLBATT] NOTIFY event. Line 524 say that this happens every RBWARNTIME = 43200 seconds (12 hours).

Line 525: Daemon upsmon will trigger a [NOCOMM] NOTIFY event after NOCOMMWARNTIME seconds if it can't reach any of the UPS entries in configuration file upsmon.conf. It keeps warning you until the situation is fixed.

Line 526: When running in primary mode, upsmon waits this long after sending the [SHUTDOWN] NOTIFY event to warn the users. After the timer elapses, it then runs your SHUTDOWNCMD as specified on line 498. If you need to let your users do something in between those events, increase this number. Don't make this too big, even though the battery still has charge. Alternatively, you can set this very low so you don't wait around when it's time to shut down.

For lots and lots of details, see man upsmon.conf. See also the file config-notes.txt in the distribution.

8.6 Configuration file upssched.conf for mgmt

Daemon upsmon in mgmt detects the NOTIFY events due to status changes in gold and in mgmt, and for those flagged as EXEC in upsmon.conf calls upssched as indicated by the NOTIFYCMD directive. The program upssched provides a richer set of actions than upsmon, especially the management of timers.

On line 528 CMDSCRIPT points to a user script to be called for designated NOTIFY events. This script will receive as argument the user chosen timer name.

Line 530 defines PIPEFN which is the file name of a socket used for communication between upsmon and upssched. It is important that the directory be accessible to NUT software and nothing else. For line 530 the Debian distribution uses /var/run/nut/upssched.pipe.

Daemon upsmon requires the LOCKFN declaration on line 531 to avoid race conditions. The directory should be the same as PIPEFN.

8.6.1 UPS-3 on gold

Lines 533 and 534 say what is to be done by upssched for a NOTIFY event [ONBATT] due to UPS-3 on gold. On line 533 the START-TIMER says that upssched is to create and manage a timer called "UPS-3-two-minute-warning-timer" which runs for 5 seconds. When this timer completes, upssched calls the user script specified by CMDSCRIPT with argument "UPS-3-two-minute-warning-timer". Line 534 does a similar thing for the 125 second timer "UPS-3-shutdown-timer".

Hopefully the back-up generator starts, and power returns before 2 minutes have gone by. Lines 535-537 say what is to be done by upssched for NOTIFY event [ONLINE]. The CANCEL-TIMER

```
527 # upssched.conf  -- mgmt --
528 CMDSCRIPT /usr/bin/upssched-cmd
529 # PIPEFN LOCKFN suitable for Debian 11
530 PIPEFN /run/nut/upssched.pipe
531 LOCKFN /run/nut/upssched.lock
532
533 AT ONBATT UPS-3@gold      START-TIMER UPS-3-two-minute-warning-timer 5
534 AT ONBATT UPS-3@gold      START-TIMER UPS-3-shutdown-timer 125
535 AT ONLINE UPS-3@gold      CANCEL-TIMER UPS-3-two-minute-warning-timer
536 AT ONLINE UPS-3@gold      CANCEL-TIMER UPS-3-shutdown-timer
537 AT ONLINE UPS-3@gold      EXECUTE UPS-3-back-on-line
538
539 AT ONBATT UPS-2@gold      START-TIMER UPS-2-two-minute-warning-timer 5
540 AT ONBATT UPS-2@gold      START-TIMER UPS-2-shutdown-timer 125
541 AT ONLINE UPS-2@gold      CANCEL-TIMER UPS-2-two-minute-warning-timer
542 AT ONLINE UPS-2@gold      CANCEL-TIMER UPS-2-shutdown-timer
543 AT ONLINE UPS-2@gold      EXECUTE UPS-2-back-on-line
544
545 AT ONBATT UPS-1@localhost START-TIMER UPS-1-two-minute-warning-timer 5
546 AT ONBATT UPS-1@localhost START-TIMER UPS-1-shutdown-timer 125
547 AT ONLINE UPS-1@localhost CANCEL-TIMER UPS-1-two-minute-warning-timer
548 AT ONLINE UPS-1@localhost CANCEL-TIMER UPS-1-shutdown-timer
549 AT ONLINE UPS-1@localhost EXECUTE UPS-1-back-on-line
550
551 AT ONBATT heartbeat@localhost CANCEL-TIMER heartbeat-failure-timer
552 AT ONBATT heartbeat@localhost START-TIMER  heartbeat-failure-timer 660
```

Figure 77: Configuration file upssched.conf for mgmt.

declarations say that upssched must cancel the timers "UPS-3-two-minute-warning-timer" and "UPS-3-shutdown-timer". The user script is not called.

Line 537 command EXECUTE says that upssched is to call the user script immediately with the argument "UPS-3-back-on-line".

8.6.2 UPS-2 on gold

UPS-2 on gold is handled in exactly the same way as UPS-3. Lines 539 and 540 define the timers which start when upssched receives a NOTIFY event [ONBATT], and lines 541 and 542 cancel those timers when hopefully upssched receives NOTIFY event [ONLINE].

Line 543 command EXECUTE says that upssched is to call the user script immediately with the argument "UPS-2-back-on-line".

8.6.3 UPS-1 on mgmt

UPS-1 on mgmt is also handled in exactly the same way as UPS-3. Lines 545 and 546 define the timers which start when upssched receives a NOTIFY event [ONBATT], and lines 547 and 548 cancel those timers when hopefully upssched receives NOTIFY event [ONLINE], however if power does not return before two minutes have gone by, the timer "UPS-1-shutdown-timer" will complete and upssched will call the user script with the parameter "UPS-1-shutdown-timer" .

Line 549 command EXECUTE says that upssched is to call the user script immediately with the argument "UPS-1-back-on-line".

8.6.4 heartbeat on mgmt

On line 551, when daemon upssched receives an [ONBATT] it executes the command CANCEL-TIMER heartbeat-failure-timer. This kills the timer. upssched does not call the user script.

Immediately afterwards, on line 552, and for the same [ONBATT] event, upssched executes command START-TIMER heartbeat-failure-timer 660 which restarts the heartbeat-failure-timer which will run for another 660 sec, i.e. 11 minutes. If the timer completes, upssched will call the user script upssched-cmd with parameter "heartbeat-failure-timer".

8.7 User script upssched-cmd

```
553 #!/bin/bash -u
554 # upssched-cmd  -- mgmt --
555 logger -i -t upssched-cmd Calling upssched-cmd $1
556
557 # Send emails to/from these addresses
558 EMAIL_TO="sysadmin@example.com"
559 EMAIL_FROM="upssched-cmd@${HOSTNAME:-nut}.example.com"
560
561 function make-STCH {
562    STCH="[$( upsc $1 ups.status )]:$( upsc $1 battery.charge )%"}
563 case $1 in
```

Figure 78: User script upssched-cmd on mgmt, 1 of 5.

The user script upssched-cmd, the example we show is in Bash, manages the completion of UPS-3-two-minute-warning-timer, UPS-2-two-minute-warning-timer, UPS-1-two-minute-warning-timer, UPS-3-shutdown-timer, UPS-2-shutdown-timer, UPS-1-shutdown-timer, UPS-3-back-on-line, UPS-2-back-on-line, UPS-1-back-on-line and heartbeat-failure-timer.

There is no such thing as a single script which fits all industrial situations, but here is an example of what can be done. You will probably need to modify this for your own use. Note that this script could be written in the language of your choice, as long as the resulting program is able to receive

the timer names as a parameter, send e-mails and log and notify the users of messages. Bash has the advantage of being widely available and is understood by many sysadmins.

In figure 78, on lines 558 and 559, change the e-mail addresses to something that works for you.

Lines 561-562 declare a function which prepares a Bash variable `STCH` which gives the current UPS status and battery charge. This is to be included in messages, so we get a clearer idea of what is happening.

The bulk of the user script is a case statement beginning at line 563 covering all the possible parameter values (timer names) that the user script may expect.

```
564 (UPS-3-two-minute-warning-timer)  make-STCH UPS-3@gold
565    MSG="UPS-3: gold power failure. $STCH" ;;
566 (UPS-3-shutdown-timer)            make-STCH UPS-3@gold
567    MSG="UPS-3: gold shutdown. $STCH" ;;
568              Commands for undisclosed device shutdown, e.g. saltstack
569 (UPS-3-back-on-line)              make-STCH UPS-3@gold
570    MSG="UPS-3: power returns. $STCH"  ;;
```

```
571              Case "UPS-2" is very similar
```

Figure 79: User script upssched-cmd on mgmt, 2 of 5.

In figure 79, lines 564-570 cover the events associated with `UPS-3` on gold. When an [ONBATT] occurs the sysadmin receives `wall` and `notify` warnings that power to the undisclosed device has failed, and that unless alternative power becomes available in two minutes, the undisclosed device will be shut down. These warnings contain the text assembled in Bash variable `MSG`. Additionally, when the [ONBATT] occurs upssched begins a two minute timer `UPS-3-shutdown-timer`. If no alternative power appears, and this timer expires, the installation specific code on line 568 will shut down the undisclosed device attached to gold. This code might for example be based on the `saltstack` remote management tools.

```
572 (UPS-1-two-minute-warning-timer)  make-STCH UPS-1
573    MSG="UPS-1: gold power failure. $STCH" ;;
574 (UPS-1-shutdown-timer)            make-STCH UPS-1
575    MSG="UPS-1: gold shutdown. $STCH"
576    /usr/sbin/upsmon -c fsd ;;
577 (UPS-1-back-on-line)              make-STCH UPS-1
578    MSG="UPS-1: power returns. $STCH"  ;;
```

Figure 80: User script upssched-cmd on mgmt, 3 of 5.

In figure 80, lines 572-578 cover the events associated with `UPS-1` on mgmt. When an [ONBATT] occurs the sysadmin receives `wall` and `notify` warnings that power to the management workstation has failed, and that unless alternative power becomes available in two minutes, the workstation will be shut down. These warnings contain the text assembled in Bash variable `MSG`. Additionally, when

the [ONBATT] occurs upssched begins a two minute timer `UPS-1-shutdown-timer`. If no alternative power appears, and this timer expires, the command `upsmon -c fsd` on line 576 will shut down the workstation by executing the command specified by `SHUTDOWNCMD` on line 498.

```
579 (heartbeat-failure-timer)         make-STCH heartbeat
580    MSG="NUT heart beat fails. $STCH" ;;
581    MSG1="Hello, upssched-cmd reports NUT heartbeat has failed."
582    MSG2="Current status: $STCH \n\n$0 $1"
583    MSG3="\n\n$( ps -elf | grep -E 'ups[dms]|nut' )"
584    echo -e "$MSG1 $MSG2 $MSG3" | /bin/mail -r "$EMAIL_FROM" \
585         -s "NUT heart beat fails. Currently $CHMSG" "$EMAIL_TO" ;;
```

Figure 81: User script upssched-cmd on mgmt, 4 of 5.

In figure 81, lines 579-585 cover the event associated with `heartbeat` on mgmt. The "heartbeat" technique is discussed in detail in chapter 6. If the `heartbeat-failure-timer` completes then something is wrong with NUT, and lines 581, 582 and 583 prepare a message for the sysadmin in Bash variables `MSG1`, `MSG2` and `MSG3`. Lines 584-585 e-mail the message to the sysadmin. The message includes the current state of those NUT kernel processes which are operational.

```
586 (*) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
587       exit 1 ;;
588 esac
589 logger -i -t upssched-cmd $MSG
590 notify-send-all "$MSG"
```

Figure 82: User script upssched-cmd on mgmt, 5 of 5.

In figure 82, lines 586-587 cover any unexpected parameter values, and lines 589-590 log the message and pass it to the system notification.

8.8 The shutdown story

UPS-3 on gold: If UPS-3 detects that power has failed, and takes over the supply to the undisclosed device, then the NUT setup will advise the system administrator on the mgmt workstation. If the backup generator comes on automatically before two minutes, then the sysadmin on mgmt will be informed, but if power does not re-appear, then script upssched-cmd in mgmt will remotely command the "shutdown" of the undisclosed device. A complete shutdown may be impossible, and all that can be done for some equipment is to put it into a quiescent state. The management workstation mgmt is not shut down.

UPS-2 on gold: If UPS-2 detects that its own power supply has failed, and that it is now powering gold, then the NUT setup of this chapter will advise the system administrator on the mgmt workstation. With the example configuration, if power is not restored in two minutes then an action in the script upssched-cmd will shut down both gold and the undisclosed device. Workstation mgmt is not shut down.

UPS-1 on mgmt: If UPS-1 detects that its own power supply has failed, and the workstation management is now on battery power, then we enter the scenario described in detail in chapter 7. There is no need to shutdown the undisclosed device or gold. A backup workstation on a different site could take over the management of UPS-3 and UPS-2.



Part 2

TLS support for upsd and clients

This part describes TLS support for current release 2.8.0 and the previous release 2.7.4 which is still part of current distributions.

If you are not interested in Python internals, then one click and you are in chapter 10.

9 Introduction

The description of the Python3 scripts in this Part supposes that you have some experience as a system administrator and that you are already familiar with NUT, it's component daemons and configuration files as described in Part 1.

9.1 Use of Python3

9.1.1 No object orientation

The Python language was originally designed in the apparent belief that all would be OO, but this is now weakening[24] *as one writer put it « in order to attract a larger audience ».*

The Python3 scripts are not "object oriented" (OO). NUT itself is a process control application and is "event oriented", not "object oriented". The Python scripts of part 2 are similarly "event oriented", and the design will be evident to those familiar with the NUT C code.

The Python scripts proposed for NUT provide a set of functions, and a main program written in an imperative style — very similar to the NUT C programs. The coding syntax itself is influenced by the OO origins of Python. For example the concatenation of two strings `a` and `b` is written `''.join([a, b])`. In OO parlance the class of the empty string `''` provides the method `join` with a list of parameters. However no OO skill or conviction is needed to read the proposed scripts.

9.1.2 Lint-free code

The Python3 scripts described in this documentation are "lint free" as determined by the pylint program which follows the PEP 8 style guide for Python code. Since the Python3 programs described here are a contribution to NUT rather than the general Python ecosystem, changes have been made

[24] See Object-Oriented Programming — The Trillion Dollar Disaster, Ilya Suzdalnitski.

	Global changes from default in pylintrc
1	Indentation string reduced from 4 to 2 spaces.
2	Allow lines up to 132 characters instead of 100.
3	Disabled the undefined-variable option. This looks like a pylint bug.
4	Disabled the bad-whitespace, bad-continuation, multiple-statements and broad-except options.
5	Removed statistical reports from output.
6	Comment out the deprecated option "symbols".
7	Option include-naming-hint is turned on.
8	Option max-module-lines increased from 1000 to 4000.
9	Options module-rgx and module-naming-hint changed. Modules may have names of form `[a-zA-Z][a-zA-Z0-9]*`
10	Option variable-rgx allows uppercase letters. Variables may have names of the form `[a-zA-Z_][a-zA-Z0-9_]{2,30}`
11	Option const-rgx allows lower case letters. Constants may have names of the form `([a-zA-Z_][a-zA-Z0-9_]*)|(__.*__)`

	Local changes from default included in code
12	`# pylint: disable=global-statement` Python PEP 8 dislikes the use of global variables. We find simple and effective use, and inhibit the warning.
13	`# pylint: disable=anomalous-backslash-in-string` This warning is a Pylint false positive.
14	`# pylint: disable=undefined-loop-variable` Pylint dislikes `var = var + ...`

Figure 83: File pylintrc, Changes to the default Python style.

to allow NUT characteristics to be freely expressed. These changes to the default Python style are defined by file pylintrc, and shown in figure 83.

The PEP 8 style guide for Python code requires that no line include trailing spaces. To remove trailing spaces using emacs, try command `M-x replace-regexp RET ␣+$ RET RET` where ␣ is a space. How does vim do this? The l33t use commnd `:%s/\s\+$//e`

10 mkNUTcert.py builds TLS certificates

A secure network connection between the Attachment Daemon and the Management Daemon requires use of TLS (Transport Layer Security) public and private keys. TLS replaces its now-deprecated predecessor, Secure Sockets Layer (SSL) used by release 2.7.4 upsd and upsmon. Building keys which meet the increasingly complex requirements of the Internet is not obvious. The Python3 utility script mkNUTcert.py described here builds a TLS private key for a server such as upsd, a self-signed CA certificate and a certificate for a client such as upsmon that wishes to access the server. The status is "experimental". The script is optimised for use with NUT and is expected to be run on the same machine as upsd. It is intended for demonstration and experiment. The license is GPL v3 or later at your choice, with support in the `nut-upsuser` mailing list.

10.1 Very Short Introduction to TLS Certificates

SSL and the TLS that has replaced SSL are a quagmire of technical terms many of which are out-of-date, confusing or incorrectly used. The OpenSSL project has produced a Swiss Army Knife[25] of utilities which are the best known tools for work in this area. Anyone venturing into this mess has to do a lot of reading. Here is a very short list.

- The Network UPS Tools User Manual, chapter 9, Notes on securing NUT.
- The NUT man pages `man upsd.conf` and `man upsmon.conf`.
- The command `openssl help` followed by `openssl` *`command`* `-help` for details of the options offered by the *`command`* tool.
- The openssl man page and it's copious "See Also".
- Ivan Ristić's "A Short Guide to the Most Frequently Used OpenSSL Features and Commands" available at web site feistyduck.com OpenSSL Cookbook.
- Web site digitalocean.com, OpenSSL Essentials: Working with SSL Certificates, Private Keys and CSRs.
- Web site zytrax.com, Survival guides - TLS/SSL and SSL (X.509) Certificates.
- Website how2ssl.com, OpenSSL tips and common commands.

Here is a short summary of technical terms used in this chapter, see also this post.

Certificate A file containing the public key used by clients to communicate with the server, possibly with additional information. For public keys we use file names of the form *`mybox`*`-client.cert.pem` where *mybox* is the name of the upsd server.

[25] I counted 48 tools in version 1.1.0f.

Certificate Authority (CA) Commercial businesses and others who want their customers to feel safe using their sites have their TLS certificates verified by a Certificate Authority (CA). You apply with a Certificate Signing Request (CSR), pay and receive a copy of your certificate linked to a trusted root certificate, for some meaning of "trust".

Where does NUT stand? We are our own Certificate Authority and the certificate we create is itself the root certificate. We do nut use CSRs. We trust ourselves. In a closed industrial context where few people have access to the systems, this provides better security than the commercial offerings used on the web. Quoting from RFC 5280, section 3.2:

> (a) Certification paths start with a public key of a CA in a user's own domain, or with the public key of the top of a hierarchy. Starting with the public key of a CA in a user's own domain has certain advantages. In some environments, the local domain is the most trusted.

Root certificate A Certifying Authority takes the private key and provides a certificate of authenticity known as a "root certificate". However in the commercial world intermediaries appear and get paid to add their certificates, thus forming a "chain of trust". NUT does not have such a chain. The root certificate is the only one. In NUT's self-signed world, the upsd server uses as private key a file which contains the private key and then the root certificate[26]. For the private key we use a file name of the form *mybox*`.cert.pem` where *mybox* is the name of the upsd server. The clients will use just the root certificate which contains the public key.

PEM PEM is an encoding [27] format for a certificate which is already ASN1 encoded and which allows it to be included in "ascii" base 64 files. If you are curious, the three letters PEM stand for Privacy-enhanced Electronic Mail. We use file type `.cert.pem` for these certificate files, but you will also find such certificates with just the `pem` extension.

CSR A Certificate Signing Request contains the private key and the additional information needed to build the public key certificate. A CSR is needed for public sites for which an expensive external service will sign the certificate as authentic and valid (for some value of authentic and valid). Since UPS units are not a public matter, we sign our own certificates. NUT does not use CSR's.

[26] In that order. See figure 85

[27] Historically, this encoding was used for early networks which only guaranteed to transmit 7 of the 8 bits in a byte.

10.2 Overview of mkNUTcert.py

The script has many options, but in general few and in some simple cases none at all are needed. To see the options and their default values enter command `mkNUTcert.py --help`

```
$ mkNUTcert.py --help
usage: mkNUTcert.py  [-h] [-SAN <list of server names>]
  [-C <ISO 3166 two letters>] [-O <name>] [-OU <unit name>]
  [--serialNumber <integer>] [--notBefore <integer>]
  [--notAfter <integer>] [-s <filename>] [-c <filename>] [-v]
```

Figure 84: Command `mkNETcert.py --help`.

Let's look at these optional arguments in more detail.

`--clientcertfile` *<filename>*, `-c` *<filename>* File path and name for the client's certificate. mkNUTcert.py tries to guess where to put things. Lucky Debian users might see `/etc/nut/`*mybox*`-client.cert.pem` . All the clients of the upsd server use this certificate.

`--countryName` *<ISO 3166 two letters>*, `-C` *<ISO 3166 two letters>* Please feel free to specify your 2 digit ISO 3166 Country Codes. The default is "`FR`".

`-h, --help` show this help message and exit

`-O` *<name>*, `--organisationName` *<name>* The proud default for organisation name is "`Network UPS Tools`". You probably don't have to change this.

`-OU` *<unit name>*, `--organisationUnitName` *<unit name>* The default value for the Organisation Unit name is "`mkNUTcert.py version 1.1`". Again, you probably don't have to change this.

`--serialNumber` *<integer>* The default for the serial number is `1`.

`--servercertfile` *<filename>*, `-s` *<filename>* File path and name for the server's certificate. mkNUTcert.py tries to guess where to put things. Lucky users of Debian might see `/etc/nut/`*mybox*`.cert.pem` . See table 104 for a list of possible directories.

`--subjectAltName` *<list of server names>*, `-SAN` *<list of server names>* You may well want to change this option. It defines a space separated list of names of the upsd server. The default is "*mybox* `localhost 10.218.0.19 mybox.example.com`" where *mybox* is the name of the machine on which you have run mkNUTcert.py. In earlier releases of SSL/TLS the option CN (Common Name) was used to specify the server name. This is now deprecated in favour of SAN (subjectAltName).

`--notAfter <integer>` The validity end time in seconds from now. The default is `0`, i.e. indefinite validity. Note that the value specified in the certificate is `Dec 31 23:59:59 9999 GMT` as required by RFC 5280 para 4.1.2.5.

`--notBefore <integer>` The validity start time is seconds from the moment you run the program. The default is `0`, i.e. now. You probably don't have to change this.

`-v, --version` Show mkNUTcert.py, Python and SSL/TLS versions, then exit.

10.3 What mkNUTcert.py provides

The private key and public keys, known as certificates) provided by mkNUTcert.py are in the form of PEM encoded files:

- Root certificate *mybox*`.cert.pem`
- Public certificate *mybox*`-client.key.pem`

10.3.1 Private Key and Certificate = Root Certificate

The server's root certificate, i.e. private key with a self-signed certificate, PEM encoding can be seen with command shown on line 596 in figure 85:

This file, like the root certificate, should be protected. It should have very restricted ownership and permissions.

```
596 $ grep -A1 -E "^---" /etc/nut/titan.cert.pem
597 -----BEGIN PRIVATE KEY-----
598 MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQC2sJigLVujiJ0/
599 --
600 -----END PRIVATE KEY-----
601 -----BEGIN CERTIFICATE-----
602 MIIFhDCCA2ygAwIBAgIBATANBgkqhkiG9w0BAQ0FADBMMQswCQYDVQQGEwJGUjEa
603 --
604 -----END CERTIFICATE-----
```

Figure 85: Root certificate = private key and certificate.

If you attempt to display the contents of the root certificate using the command. `openssl x509 -text -noout -in /etc/nut/titan.cert.pem` then only the certificate is displayed, as shown in figure 87.

10.3.2 Public Key Certificate

The client's public key certificate contains the public key and certifies[28] that it is indeed the public key corresponding to the upsd server's private key. It contains only a `CERTIFICATE` part, not the `PRIVATE KEY` part. The PEM encoding can be seen with command shown on line 605 in figure 86:

```
605 root@titan ~ grep -A1 -E "^---" /etc/nut/titan-client.cert.pem
606 -----BEGIN CERTIFICATE-----
607 MIIFhDCCA2ygAwIBAgIBATANBgkqhkiG9w0BAQ0FADBMMQswCQYDVQQGEwJGUjEa
608 --
609 -----END CERTIFICATE-----
```

Figure 86: The client's PEM encoded public certificate.

Details of the certificate can be seen with the command shown on line 610 in figure 87 which shows a self-signed public certificate:

1. The certificate is certified directly by the server's root certificate and there are no intermediate certificates. NUT acts as it's own certifying authority. For tightly controlled situations such as UPS management, this provides better security.
2. The certificate is self-signed. The issuer on line 616 is also the subject on line 620 as required by RFC 5280 para 4.1.2.4 last sentence.
3. The value "`Dec 31 23:59:59 9999 GMT`" on line 619 is defined by RFC 5280 para 4.1.2.5.
4. The public key begins on line 625.
5. There is no Authority Key Identifier which is obligatory for Web certificates. This omission is specific to self-signed certificates, see RFC 5280 para 4.2.1.1.

[28] A public key certificate provides a safe way for an entity to pass on its public key to be used in asymmetric cryptography. The public key certificate avoids the following situation: if Charlie creates his own public key and private key, he can claim that he is Alice and send his public key to Bob. See techtarget.com.

```
root@titan ~ openssl x509 -text -noout -in /etc/nut/titan-client.cert.pem
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: sha512WithRSAEncryption
    Issuer: C = FR, O = Network UPS Tools, OU = mkNUTcert.py version 1.0
    Validity
      Not Before: Oct 22 10:55:53 2020 GMT
      Not After : Dec 31 23:59:59 9999 GMT
    Subject: C = FR, O = Network UPS Tools, OU = mkNUTcert.py version 1.0
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        RSA Public-Key: (4096 bit)
        Modulus:
          00:c0:91:c2:1c:68:83:b7:83:1e:c7:89:45:1e:c4:
          ...
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints: critical
        CA:TRUE
      X509v3 Subject Alternative Name:
        DNS:titan, DNS:localhost, DNS:10.218.0.19, DNS:titan.example.com
      X509v3 Subject Key Identifier:
        DA:39:A3:EE:5E:6B:4B:0D:32:55:BF:EF:95:60:18:90:AF:D8:07:09
  Signature Algorithm: sha512WithRSAEncryption
     8c:39:6a:dc:74:41:65:de:c6:e2:0d:68:1e:61:bf:8f:d7:56:
     ...
```

Figure 87: The self-signed public certificate.

10.4 Running mkNUTcert.py

1. Before running the script, check the shebang `#!` in the first line. The default value is `#!/usr/bin/python3 -u` . Check that you have a sufficiently recent version of Python3 at that address. If your version is not sufficiently recent, you will receive an error message from mkNUTcert.py. How do I know if I have a sufficiently recent version of Python3? Try running the script. If it runs, you're ok. Otherwise you will need to upgrade your Python installation.
2. Run command `mkNUTcert.py --help` to see the default values. Pay extra attention to the following:

Option	Default Value
`--subjectAltName`	*host* `localhost 10.218.0.19` *host*`.example.com`
`--servercertfile`	*Script-tries-to-guess*`/`*host*`.cert.pem`
`--clientcertfile`	*Script-tries-to-guess*`/`*host*`.client.cert.pem`

   The script also attempts to guess the owner:group for the two output files. You should review that choice.
3. When you run the command `mkNUTcert.py` you will be reminded of the proposed file paths and file names for the certificates. Enter "`yes`" to confirm and anything else to exit immediately. If you continue, mkNUTcert.py will report:

```
638 Writing private key with self-signed certificate for server to file ...
639 This file must be protected.  E.g. do not make it world readable.
640 Current owner is nut:nut with permissions 0o600.
641
642 Writing user certificate for client to file ...
643 The user (i.e. client) certificate should be installed in all monitors.
644 Current owner is nut:nut with permissions 0o644.
```

4. Ensure that the private key and the root certificate are properly protected. Only root and the user designated to run upsd should have access to the private key. No-one else.

   The root certificate is given restrictive permissions `600`. If you attempt to run the script a second time it may well refuse if there is already a root certificate at the same address with such restrictive permissions. You have to remove the old root certificate yourself as user root. Take care!

   The user (i.e. the client) certificate is given permissions `644`.
5. Transfer the user certificate to the machine(s) running the Management Daemon, e.g. upsmon. Check that ownership and permissions are correct on the destination machine.

11 Encrypted connections

The configurations we have seen so far assume that the connection between the NUT client and the NUT server is either in the same machine or over a local, well protected network. The client's password is transmitted in clear text to the server. This may be a reasonable risk locally, but is not acceptable if client and server are connected by a public network or by a network deemed to be at risk. This chapter looks at the technique for encrypting the traffic between client and server made possible by TLS 1.3 support in NUT 2.8.0.

Chapter 12 discusses the use of TLS shims to provide the same encryption for NUT 2.7.4.

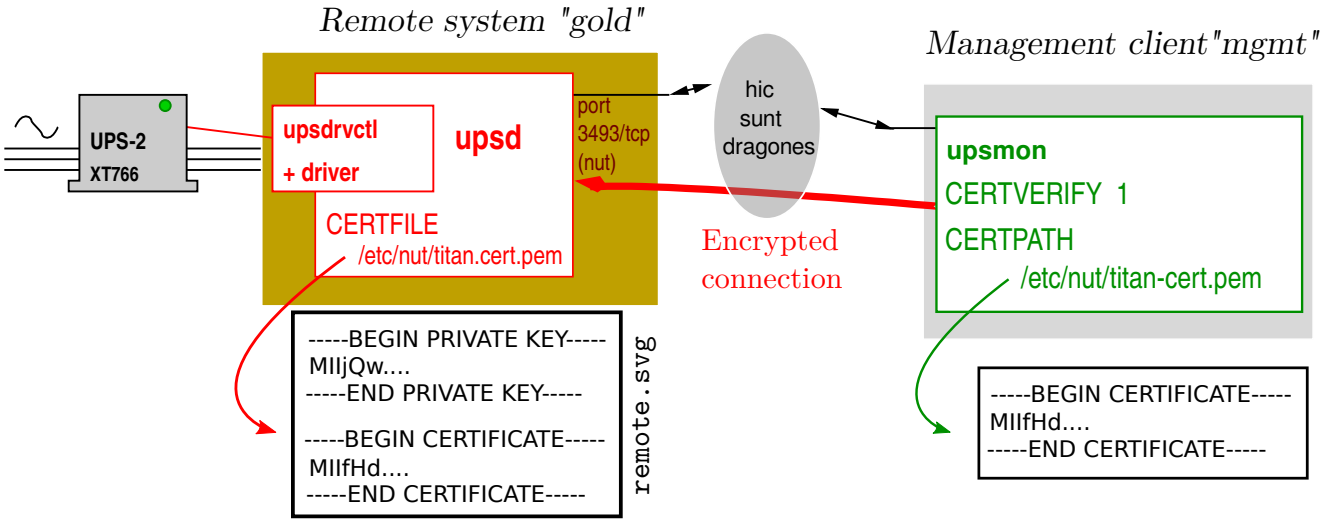


Figure 88: Encrypted connection to remote server.

See chapter 10.1 for a very short introduction to the quagmire of technical terms many of which are confusing or incorrectly used.

This chapter will continue the habit of chapter 8 of referring to the server to which the UPS is connected as gold and the management client as mgmt.

11.1 Additional configuration files

In addition to the configuration files discussed in previous chapters, the following configuration files are also needed for encrypted communication between remote NUT server gold and management client mgmt.

11.1.1 In the remote server "gold"

gold: Add the following lines to upsd.conf. See man upsd.conf

```
645 # upsd.conf
646 ...
647 DISABLE_WEAK_SSL true
648 CERTFILE /etc/nut/titan.cert.pem
```

Line 647 prevents use of insecure early versions of SSL/TLS by restricting upsd to use TLSv1.2 or better.

On line 648 the upsd daemon access control upsd.conf needs the private key generated by mkNUTcert.py. The CERTFILE[29] declaration declares the file containing the root certificate, i.e. the private key and the certificate in PEM format. See chapter 10.3.1.

11.1.2 In each management client "mgmt"

mgmt: Add the following lines to upsmon.conf. See man upsmon.conf

```
649 # upsmon.conf
650 ...
651 CERTVERIFY 1
652 CERTPATH /etc/nut/titan-client.cert.pem
```

Line 651 makes upsmon verify all connections with certificates. 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.

In line 652 CERTPATH points to a file containing a certificate in PEM format, used to verify the server certificate presented by the upsd server.

[29] The name "CERTFILE" is a poor choice since it is a private key not a public key. A name such as "KEYFILE" would have been better. Normally it is public keys that are referred to as "certificates".

11.2 Debugging: Sniffing port 3493

Testing is essential to achieve the required level of security, and a key part of this testing is sniffing the network to ensure that the connections to port 3493 on the NUT server gold are indeed encrypted.

We use tcpdump on Debian for this testing. Other network sniffing software is available. The first test is to see the clear text nature of the non-encrypted communication.

1. In the server, gold, or in the management client mgmt, run the command `tcpdump -A port nut` as root.
2. In the management client mgmt, stop upsmon, and then restart it with the command `systemctl restart nut-monitor.service`.
3. `tcpdump` will display the trace shown in figure 89 which has been edited to make it easier to read. Line 657 shows the client mgmt attempting to begin an encrypted session which is refused by server gold on line 659. Line 663 shows the password transmitted in clear text. *Let this be a warning to you.*

   Lines 669-672: Client mgmt then makes a plain text request every 5 seconds for the status of `UPS-3` which the server gold then answers in plain text.

```
653 listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes
654 IP mgmt.33656 > gold.nut:
655 IP gold.nut > mgmt.33656:
656 IP mgmt.33656 > gold.nut:
657 IP mgmt.33656 > gold.nut: STARTTLS
658 IP gold.nut > mgmt.33656:
659 IP gold.nut > mgmt.33656: ERR FEATURE-NOT-CONFIGURED
660 IP mgmt.33656 > gold.nut:
661 IP mgmt.33656 > gold.nut: USERNAME upsmaster
662 IP gold.nut > mgmt.33656: OK
663 IP mgmt.33656 > gold.nut: PASSWORD sekret
664 IP gold.nut > mgmt.33656: OK
665 IP mgmt.33656 > gold.nut: LOGIN UPS-3
666 IP gold.nut > mgmt.33656: OK
667 IP mgmt.33656 > gold.nut: MASTER UPS-3
668 IP gold.nut > mgmt.33656: OK MASTER-GRANTED
669 IP mgmt.33656 > gold.nut: GET VAR UPS-3 ups.status
670 IP gold.nut > mgmt.33656: VAR UPS-3 ups.status "OL"
671 IP mgmt.33658 > gold.nut:
672 IP mgmt.33656 > gold.nut: GET VAR UPS-3 ups.status
673 IP gold.nut > mgmt.33656: VAR UPS-3 ups.status "OL"
```

Figure 89: tcpdump of `systemctl start nut-monitor.service` without encryption.

11.3 Testing the TLS setup

This test was done using a hybrid setup in which a version 2.8.0 upsmon talks to a version 2.7.4 upsd equipped with a shim upsdTLS.py. As shown in figure 92 upsd listens on customary port 3493 (nut), but the shim is listening on port 401.

First we trace the unencrypted traffic on port 3493 (nut). The trace has been edited to make it easier to read:

```
674 ~ tcpdump -i any port 3493 -c 2 -A
675 ...
676 IP mgmt.53634 > gold.nut: GET VAR UPS-3 ups.status
677 IP gold.nut > mgmt.53634: VAR UPS-3 ups.status "OL"
```

Figure 90: Unencrypted traffic on port 3493 (nut).

And now the same message exchange but on port 401:

```
678 ~ tcpdump -i any port 401 -c 2 -A
679 ...
680 IP mgmt.41248 > gold.401: *..'5Q.W.2..U.&..!.....i.^..-..j.....%..Q~
681 IP gold.401 > mgmt.41248: +6...&..u....6.F6.h.R.............G5..1Y
```

Figure 91: Encrypted traffic on port 401.



12 Shim daemons upsdTLS.py and upsmonTLS.py

The NUT project is now mature and proceeds at cautious speed. The SSL/TLS features of release 2.7.4 became obsolete and were deprecated before the next relase 2.8.0 appeared. The RFC 9271 proposed to address this security problem with a pair of TLS support shims sitting one beside upsd and the other in the client system.

This chapter describes an experimental implementation of the shims in of the scripts upsdTLS.py and upsmonTLS.py. The scripts and their SHA1 check sums may be downloaded from http://rogerprice.org/NUT

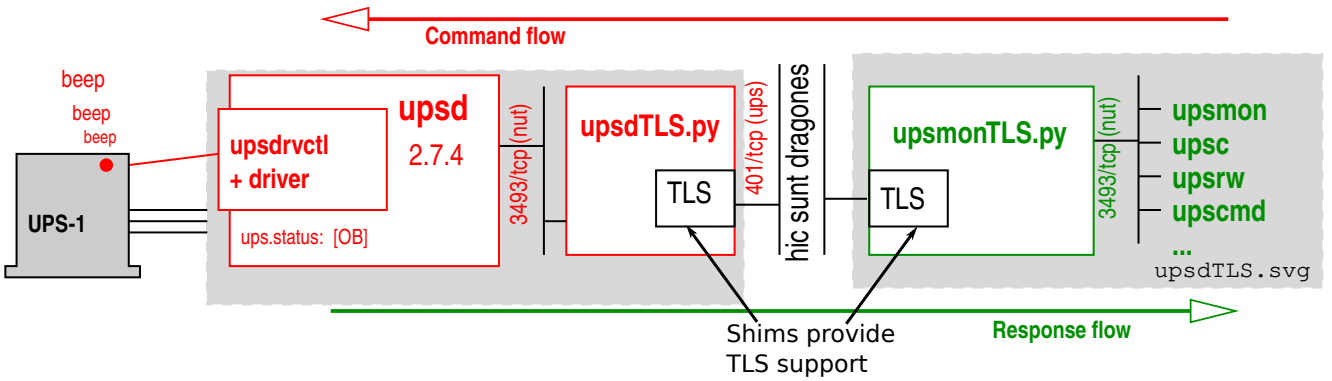


Figure 92: NUT 2.7.4 TLS support using shims upsdTLS.py and upsmonTLS.py.

NUT 2.7.4 did not support the latest versions of TLS. This prevented NUT 2.7.4 from using TLS since TLS strongly deprecates use of earlier versions which are no longer considered secure. To overcome this difficulty, Python script upsdTLS.py provides a shim to help upsd work with the latest, and most secure, versions of TLS. upsdTLS.py runs as a daemon alongside upsd receiving TLS encrypted traffic from it's companion shim upsmonTLS.py or from a TLS enabled client such as UPSmon.py and passing on that traffic to local upsd using an unencrypted socket. The script's status is "experimental", and is intended for demonstration and experiment. It must run on the same machine as upsd. The license is GPL v3 or later at your choice, with support in the `nut-upsuser` mailing list.

12.1 Overview of Shim upsdTLS.py

The script has no configuration files, but has many options. In general few and in some simple cases none at all are needed. To see the options and their default values you can enter command `upsdTLS.py --help` .

Let's look at these optional arguments in more detail. XXX

`--backlog <integer>` Maximum incoming message backlog, default value 5. You should not usually need to change this.

```
682 $ upsdTLS.py --help
683 usage: upsdTLS.py  [--backlog <integer>] [-D] [-h] [-l <file>]
684   [--listen <IPv4_address> <port_number>] [--listentimeout <float>]
685   [--maxconn <integer>] [--PIDfile<file>] [-s <file>] [-u <user>]
686   [--upsdport <integer>] [--upsdtimeout <float>] [-v]
```

Figure 93: Command `upsdTLS.py --help`

`-D, --debug` Increase the debugging level, may be repeated but then you get more than any human can read. Debugging output is written into the NUT log file.

`-h, --help` Show this help message and exit

`--listen` *<IPv4_address> <port_number>* upsdTLS.py listens to (i.e. receives commands from) shim upsmonTLS.py or a TLS enabled client on this interface and port, with the default `'127.0.0.1' 401`. Temporarily, we squat IANA 401/tcp (ups). Setting a port number $<$ 1024 requires starting the daemon as root.

`--listentimeout` *<float>* Socket timeout for exchanges on the port specified by `--listen`. The default is 5.0 seconds.

`-l` *<file>*, `--logfile` *<file>* The log file, with default `/var/log/NUT.log` . Progress and error messages and the copious stuff generated by option `-D` go into this file. See chapter E for an extension to `logrotate` to cover this file.

`--maxconn` *<integer>* Maximum number of incoming connections, the default is `10`. Strictly speaking, the maximum number of sockets the daemon process may have open, where `getconf OPEN_MAX` gives system file maximum. You should not usually need to change this.

`--PIDfile` *<file>* The child PID is written into this file, for the greater pleasure of systemd. The default for upsdTLS.py is `/run/nut/upsdTLS.pid` . Do not change this unless you know what you are doing. You should also review the systemd service unit.

`-s` *<file>*, `--servercertfile` *<file>* The file path and file name of the server's private key. upsdTLS.py tries to guess where to put things. The default on Debian systems is `/etc/nut/`*mybox*`.cert.pem` . OpenSUSE sysadmins would probably use `/etc/ups/...` See table 104 for a list of possible directories.

`-u` *<user>*, `--user` *<user>* After launch as root, run as this user. upsdTLS.py tries to guess the user. OpenSUSE admins would probably see `upsd`, whereas Debian admins would see `nut`. See table 104 for a list of possible users.

`--upsdport` *<integer>* Relay incoming commands to this upsd port, and (no surprise) the default relay port to upsd is 3493. upsd is assumed to be running on llocalhost.

`--upsdtimeout <float>` Socket timeout for exchanges with upsd. The default is 5.0 seconds.

`-v, --version` Show program, Python and SSL/TLS versions, then exit.

12.2 Overview of Shim upsmonTLS.py

```
$ upsmonTLS.py --help
usage: upsmonTLS.py [--backlog <integer>] [-c <file>] [-D] [-h]
  [--listen <IPv4_address> <port_number>] [--listentimeout <float>]
  [-l <file>] [--maxconn <integer>] [--PIDfile<file>]
  [-u <user>] [--upsdname <domain>] [--upsdport <integer>]
  [--upsdtimeout <float>] [-v]
```

Figure 94: Command `upsmonTLS.py --help`

.

The script has no configuration files, but lots of options. In general few and in some simple cases none at all are needed. To see the options and their default values you can enter command `upsmonTLS.py --help` .

Let's look at these optional arguments in more detail.

`--backlog <integer>` Maximum incoming message backlog, default value 5. You should not usually need to change this.

`-c <file>, --clientcertfile <file>` The file path and file name of the client's certificate (public key). upsmonTLS.py tries to guess where to put things. The default on Debian systems is `/etc/nut/mybox-client.cert.pem` . OpenSUSE sysadmins would probably use `/etc/ups/...` See table 104 for a list of possible directories.

`-D, --debug` Increase the debugging level, may be repeated but then you get more than any human can read. Debugging output is written into the NUT log file.

`-h, --help` Show this help message and exit

`--listen <IPv4_address>` upsmonTLS.py listens to the client such as upsmon or upsc on this interface and port, with the default `'127.0.0.1' 3493`.

`--listentimeout <float>` Socket timeout for exchanges on the port specified by `--listen`. The default is 5.0 seconds.

`-l <file>, --logfile <file>` The log file, with default `/var/log/NUT.log` . Progress and error messages and the copious stuff generated by option `-D` go into this file. See chapter E for an extension to `logrotate` to cover this file.

`--maxconn` *<integer>* Maximum number of incoming connections, the default is `10`. Strictly speaking, the maximum number of sockets the daemon process may have open, where `getconf OPEN_MAX` gives system file maximum. You should not usually need to change this.

`--PIDfile` *<file>* The child PID is written into this file, for the continuing pleasure of systemd. The default for upsmonTLS.py is `/run/nut/upsmonTLS.pid` Do not change this unless you know what you are doing. You should also review the systemd service unit.

`-u` *<user>*, `--user` *<user>* After launch as root, run as this user. upsdTLS.py tries to guess the user. OpenSUSE admins would probably see `upsd`, whereas Debian admins would see `nut`. See table 104 for a list of possible users.

`--upsdname` *<domain>* Relay incoming commands from the client to the system running the shim upsdTLS.py. For example `--updsname "bigserver.example.com"`. The default name is `localhost`.

`--upsdport` *<integer>* Relay incoming commands from upsmon, upsc, etc. to this upsd/shim port. The default relay port for upsmonTLS.py is 401 which the companion script upsdTLS.py listens to by default. Temporarily, we squat IANA 401/tcp (ups). Setting a port number $<$ 1024 requires starting the daemon as root.

`--upsdtimeout` *<float>* Socket timeout for exchanges with upsd. The default is 5.0 seconds.

`-v, --version` Show program, Python and SSL/TLS versions, then exit.



12.3 Summary of shims upsdTLS.py and upsmonTLS.py

upsdTLS.py and upsmonTLS.py	
`--backlog <integer>`	5
`--debug`	
`--help`	
`--listentimeout <float>`	5.0 secs
`--logfile <file>`	`/var/log/NUT.log`
`--maxconn <integer>`	10
`--upsdtimeout <float>`	5.0 secs
`--user <user>`	Debian: `nut`
`--version`	
upsdTLS.py only	
`--listen <IPv4> <port>`	127.0.0.1 401
`--PIDfile <file>`	`/run/nut/upsdTLS.pid`
`--servercertfile <file>`	`/etc/nut/mybox.cert.pem`
`--upsdport <port>`	3493
upsmonTLS.py only	
`--clientcertfile <file>`	`/etc/nut/mybox-client.cert.pem`
`--listen <IPv4> <port>`	127.0.0.1 3493
`--PIDfile <file>`	`/run/nut/upsmonTLS.pid`
`--upsdname <domain>`	`localhost`
`--upsdport <port>`	401

Figure 95: Summary of upsdTLS.py and upsmonTLS.py options and default values.

12.4 Running the shims upsdTLS.py and upsmonTLS.py

The daemons upsdTLS.py and upsmonTLS.py usually start with user root and fork to run as the same user as upsd.

If you use systemd to manage your boxes, then you will need to create new service units, since systemd is unable to start two forking services from the same unit. See `man systemd.service(5)`. There can only be one `Type=forking` per unit.

In the box running upsd create a new file by copying the service unit file `/usr/lib/systemd/system/nut-server.service` to `/etc/systemd/system/nut-py-server-shim.service` and modify the new file as shown in figure 96. where lines 694-696 and 698-699 have been changed.

```
693 [Unit]
694 Description=Network UPS Tools - nut-server TLS shim support daemon
695 After=local-fs.target network.target nut-server.service
696 Before=nut-py-client.service

697 [Service]
698 ExecStart=/usr/sbin/upsdTLS.py
699 PIDfile=/run/nut/upsdTLS.pid
700 Type=forking

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

Figure 96: systemd service unit `nut-py-server-shim.service` for upsdTLS.py.

In the box running upsmon copy the service unit file `/usr/lib/systemd/system/nut-monitor.service` to `/etc/systemd/system/nut-py-client-shim.service` and modify the new file as shown in figure 97 where lines 704-706 and 708-709 have been changed.

The `PIDfile` declarations are there to help systemd find the daemon since upsdTLS.py and upsmonTLS.py do not keep the parent process running when they fork. Note that systemd service units in `/etc` take precedence over those in `/usr/lib`. See `man systemd.unit(5)`.

```
703 [Unit]
704 Description=Network UPS Tools - TLS shim support daemon for nut clients
705 After=local-fs.target network.target nut-server.service\
        nut-py-server-shim.service
706 Before=nut-client.service

707 [Service]
708 ExecStart=/usr/sbin/upsmonTLS.py
709 PIDfile=/run/nut/upsmonTLS.pid
710 Type=forking

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

Figure 97: systemd service unit `nut-py-client-shim.service` for upsmonTLS.py.

You may choose to place the upsdTLS.py and upsmonTLS.py scripts in directory `/usr/sbin` or make `/usr/sbin/upsdTLS.py` and `/usr/sbin/upsmonTLS.py` links to wherever you put the Python scripts. After you have made the changes, you should run the command `systemctl daemon-reload` See `man systemctl(1)`.

12.4.1 Enabling the shims upsdTLS.py and upsmonTLS.py

Before running the shims the first time, you will need to run the command

```
systemctl enable nut-py-server-shim.service nut-py-client-shim.service
```

The following `systemctl` commands will be of use to you:

- `systemctl daemon-reload`
  to make any changes to the service unit available to systemd.
- `systemctl enable nut-py-server-shim.service`
  `systemctl enable nut-py-client-shim.service`
  to make the daemons upsdTLS.py and upsmonTLS.py operational and "startable".
- `systemctl start nut-py-server-shim.service`
  `systemctl start nut-py-client-shim.service`
  to start upsdTLS.py and upsmonTLS.py. Note that this will not erase the log file. If you want to clear the log file then you need to do that yourself. See also chapter E for a discussion of log rotation.
- `systemctl status nut-py-server-shim.service`
  `systemctl status nut-py-client-shim.service`
  to see the current status of the shims.
- `systemctl stop nut-py-server-shim.service`
  `systemctl stop nut-py-client-shim.service`
  to stop upsdTLS.py and upsmonTLS.py.

upsdTLS.py and upsmonTLS.py should start automatically when the system starts, but they can also be stopped and started manually with the `systemctl` commands.

Serious errors will prevent the shims from starting and you can read about them in the NUT log and in the system log. After starting the shims, check the NUT log for warnings and other error messages.

12.4.2 Listing the systemd activity

During the debugging of the shims, I saw a summary of the NUT systemd service unit activity on a Debian 11 (NUT 2.7.4) system with the command:

```
root@titan ~ systemctl list-unit-files | grep -i -E "nut|UPS|VENDOR"
UNIT FILE                                  STATE           VENDOR PRESET
nut-driver-enumerator.path                 disabled        enabled
nut-client.service                         alias           -
nut-delayed-ups-shutdown.service           enabled         enabled
nut-driver-enumerator.service              disabled        enabled
nut-driver.service                         static          -
nut-driver@.service                        disabled        enabled
nut-monitor.service                        disabled        enabled
nut-py-client-shim.service                 enabled         enabled
nut-py-monitor.service                     enabled         enabled
nut-py-server-shim.service                 enabled         enabled
nut-server.service                         enabled         enabled
ups-monitor.service                        masked          enabled
nut-driver.target                          disabled        enabled
nut.target                                 disabled        enabled
```

Figure 98: Example of systemd service unit activity for NUT.



Part 3

Appendices

A Starting NUT

```
# nut.conf
# No spaces around the "="
MODE=standalone
```

Figure 99: Configuration file `nut.conf`.

This chapter discusses the techniques used to start the NUT software. Each distribution has it's own view of how this is to be done, so you should review the systemd service units involved and the scripts that they call.

The NUT software contains several daemons which need to be started to offer the promised NUT service. These daemons are shown in the table in figure 100.

Daemon	systemd service unit	Notes
upsd	`nut-server.service`	The central daemon which maintains the abstracted view of the UPS units.
driver	`nut-driver.service`	One or more driver daemons as specified in file `ups.conf`. This service unit is started automatically by systemd whenever upsd starts. The driver needs more time to get started?, see Manuel Wolfshant's post in the NUT mailing list.
upsmon	`nut-monitor.service`	The monitor daemon specifies what is to be done for NOTIFY events.
upssched	*none*	For activity such as the heartbeat, the timed action daemon is called by the upssched-cmd script which is specified by the `NOTIFYCMD` command in `upsmon.conf`.
TLS Shim daemons defined in Part 2		
upsdTLS.py	`nut-py-server-shim.service`	The shim daemon placed in front of upsd 2.7.4 to provide TLS support.
upsmonTLS.py	`nut-py-monitor-shim.service`	The shim placed in front of upsmon 2.7.4 to provide TLS support.

Figure 100: Daemons used by NUT.

Configuration file `nut.conf` specifies which of these daemons the operating system should start, but distributions often ignore the file. The distribution choice is normally correct for a standalone workstation protected by a single UPS, but for more complex situations, you need to review what your distribution does. See chapter 8.1 and `man nut.conf`.

Strictly speaking, this file is not for NUT, but for the process which starts NUT. The initialization process is expected to source this file to know which parts of nut are to be started. Some distributions, e.g. openSUSE, ignore `nut.conf` and start the three NUT layers driver, upsd and upsmon. They assume that `MODE=standalone`. Note that there is no space around the "=" since it is assumed that shell scripts such as Debian's `/sbin/upsd` source this file.

The possible `MODE` values are:

- `MODE=none` Indicates that NUT should not get started automatically, possibly because it is not configured or that an Integrated Power Management or some external system, is used to start up the NUT components. If you enable `nut-server.service` Debian [30] will display the message:

  upsd disabled, please adjust the configuration to your needs. Then set MODE to a suitable value in /etc/nut/nut.conf to enable it.

  Enabling `nut-monitor.service` will produce a similar message[31].
- `MODE=standalone` This is the most common situation in which line 731 in figure 99 declares that NUT should be started in the "`standalone`" mode suitable for a local only configuration, with 1 UPS protecting the local system. This implies starting the 3 NUT layers, driver, upsd and upsmon and reading their configuration files.
- `MODE=netserver` Like the standalone configuration, but may possibly need one or more specific `LISTEN` directive(s) in `upsd.conf`. Since this `MODE` is open to the network, a special care should be applied to security concerns. Debian accepts starting upsmon in this mode.
- `MODE=netclient` When only upsmon is required, possibly because there are other hosts that are more closely attached to the UPS, the `MODE` should be set to netclient. If you enable Debian's systemd service unit `nut-server.service` with this mode, then you will get the same message as for `MODE=none`.

However these alternate modes are merely wishful thinking if your distribution ignores file `nut.conf`. There are other options, see `man nut.conf`.

[30] See script `/sbin/upsd`.
[31] See script `/sbin/upsmon`.

B Stopping NUT

B.1 Delayed UPS shutdown with NUT script

We saw in chapter 2, line 45, that the upsmon.conf SHUTDOWNCMD directive specifies the command to be used to shut down the system, but what about the UPS which must keep supplying power while the system shuts down? Does the UPS also shut down?, and if so, how?

Chapter 2.5 *"The shutdown story for a simple server"* explains that somewhere in your distribution, as part of the system shutdown process, there needs to be an action to send a message to the UPS to tell it that some time later, it too will shut down. The notion of "shutdown" for a UPS unit is subtle. What shuts down is usually the supply of power to the power outlets. The UPS unit cuts off the equipment for which it provides battery backup. When this happens you may hear the audible "clunk" of the relays. The unit may also act as a power strip with surge protection, but those outlets are not covered by the protection afforded by the battery.

Note that the UPS does not shutdown at the same time as the system it protects. The UPS shutdown is **delayed**. By default the delay is 20 seconds. See line 77 if you want to change this.

The delayed UPS shutdown command may be from a shell script or a systemd service unit, but in all cases the key element is the command `upsdrvctl shutdown`.

The NUT project provides a sample script, which is to be placed in a directory of things to be done at the end of the system shutdown. This depends on the distribution.

The Debian 11 distribution places the delayed shutdown script provided by NUT and shown in figure 101 in file `/usr/lib/systemd/system-shutdown/nutshutdown` . The openSUSE distribution does the same.

```
732 #!/bin/sh
733 /usr/sbin/upsmon -K >/dev/null 2>&1 && /usr/sbin/upsdrvctl shutdown
```

Figure 101: UPS shutdown script `nutshutdown`.

On line 733 the call to upsmon with option `-K` checks the `POWERDOWNFLAG` defined by line 46. The upsmon daemon creates this file when running in primary (master) mode whenever the UPS needs to be powered off. See `man upsmon.conf` for details. If the check succeeds, we are free to call upsdrvctl to shut down the UPS's. Note that if you have multiple UPS's, the command `upsdrvctl shutdown` will shut them all down. If you have say three UPS's, `UPS-1`, `UPS-2` and `UPS-3`, and you want to shut down just `UPS-2` and `UPS-3`, then you should specify those UPS's as shown in line 735. See also `man upsdrvctl`

```
734 #!/bin/sh
735 /usr/sbin/upsmon -K >/dev/null 2>&1\
        && /usr/sbin/upsdrvctl shutdown UPS-2\
        && /usr/sbin/upsdrvctl shutdown UPS-3
```

Figure 102: UPS shutdown script `nutshutdown` for 2 of 3 UPS's.

B.2 Delayed UPS shutdown with a systemd service unit

The script provided by the NUT project in chapter B.1 is executed very late in the shutdown sequence, when it is no longer possible to log the action. If you think that power management is a critical operation and that all critical operations should be logged, then you will need to call for the delayed UPS shutdown earlier in the system shutdown sequence when logging is still possible. This can be done using the systemd service unit shown in figure 103.

```
736 # nut-delayed-ups-shutdown.service
737 [Unit]
738   Description=Initiate delayed UPS shutdown
739   Before=umount.target
740   DefaultDependencies=no
741 [Service]
742   Type=oneshot
743   ExecStart=/usr/bin/logger -t nut-delayed-ups-shutdown\
                              "upsdrvctl shutting down UPS"
744   ExecStart=/usr/sbin/upsdrvctl shutdown   # Debian
745 [Install]
746   WantedBy=final.target
```

Figure 103: UPS shutdown service unit `nut-delayed-ups-shutdown.service`.

The `ExecStart` directive on line 744 will shutdown [32] all the UPS units managed by this system. The code given is for Debian: other distributions put upsdrvctl elsewhere. If you have say three UPS's, `UPS-1`, `UPS-2` and `UPS-3`, and you want to shut down just `UPS-2` and `UPS-3`, then instead of line 744 you should specify the required UPS's as shown in lines 747-748.

```
747   ExecStart=/sbin/upsdrvctl shutdown UPS-2   # Debian
748   ExecStart=/sbin/upsdrvctl shutdown UPS-3
```

Note that this service unit does not perform the `upsmon -K` test for the `POWERDOWNFLAG`.

The position of this service unit may vary from one distribution to another, see section "unit file load path" in `man systemd.unit(5)`. For example in the openSUSE and Debian distributions, `/etc/systemd/system` is for a user's scripts, and `/usr/lib/systemd/system-shutdown` is for system scripts. You might use the `/etc/systemd/system` directory if your script is not part of an officially distributed product.

If you install or change this service unit, run command `systemctl --system reenable /etc/systemd/system/nut-delayed-ups-shutdown.service` . Maybe your distribution offers a graphical manager to do this.

For gory details see the systemd documentation. There are over 200 man pages starting with an index. For details of the directories used, see section "unit file load path" in `man systemd.unit`.

[32] The upsdrvctl program is normally a frontend to the drivers, but in the case of the `shutdown` option upsdrvctl does not use the existing driver; it creates a new driver for itself.

C Users and Directories for NUT

NUT normally runs as a non-root user, however the user varies from one distribution to another. Table 104 shows a list of users for a range of distributions. Table 104 also shows the directories used by different distributions for configuration files such as `upsd.conf`.

Distribution	ID	User: Group	Directory	ID source
Aix	aix	nut ?	/etc/nut/ ?	uname -a
Amazon	amzn	nut	/etc/ups/ ?	/etc/os-release
Arch	arch	nut	/etc/nut/	/etc/os-release
CentOS	centos	nut	/etc/ups/	/etc/os-release
Apple	darwin	nut	/etc/nut/	uname -a
Debian	debian	nut:nut	/etc/nut/	/etc/os-release
Fedora	fedora	nut	/etc/ups/	/etc/os-release
FreeBSD	freebsd	uucp	/usr/local/etc/nut/	uname -a
Gentoo	gentoo	nut	/etc/nut/	/etc/gentoo-release
HP-UX	hpux	nut ?	/etc/nut/ ?	uname -a
IPFire	ipfire	nutmon	/etc/nut/	uname -a
Kali	kali	nut	/etc/nut/	/etc/os-release
Mint	linuxmint	nut	/etc/nut/	/etc/os-release
Apple	mac	nut ?	/etc/nut/ ?	uname -a
Mageia	mageia	nut	/etc/nut/	/etc/os-release
Manjaro	manjaro	nut	/etc/nut/	/etc/os-release
NetBSD	netbsd	nut ?	/etc/nut/ ?	uname -a
Oracle	ol	nut	/etc/ups/	/etc/os-release
OpenBSD	openbsd	ups (1)	/etc/nut/	uname -a
OpenIndiana	openindiana	nut	/etc/nut/	uname -a
OpenSUSE	opensuse	upsd	/etc/ups/	/etc/os-release
Raspbian	raspbian	nut	/etc/nut/	/etc/os-release
Red Hat	rhel	nut	/etc/ups/	/etc/os-release
Slackware	slackware	nut	/etc/nut/	/etc/os-release
SUSE	sles	upsd	/etc/ups/	/etc/os-release
SUSE+SAP	sles_sap	upsd	/etc/ups/	/etc/os-release
Synology	synology	root ?	/usr/syno/etc/nut/	uname -a
Ubuntu	ubuntu	nut	/etc/nut/	/etc/os-release

The editor will be very pleased to hear of errors or omissions in this table.

Figure 104: Users and directories for NUT.

Notes:

1. The OpenBSD user may be `_ups` which is an OpenBSD convention for identifying unprivileged users. Most OpenBSD add-on software uses unprivileged usernames beginning with an underscore.
2. If NUT is built without specifying the user, then the user is `nobody:nobody`.
3. FreeNAS identifies itself in `/etc/os-release` as FreeBSD.
4. The IPFire wiki suggests user `nutmon` for upsmon but makes no mention of upsd.
5. OpenIndiana: historically, NUT was not included as a package in OpenIndiana, and an OpenIndiana Wiki entry dated 2013 recommended user `ups` and directory `/opt/nut/etc/`. The values in the table are taken from OpenIndiana's current Github data for NUT.



D Using notify-send

The program "wall" used by NUT to put notifications in front of the users is now well past it's best-before date and hardly fit for purpose. It has not been internationalized, does not support accented letters or non-latin characters, and is ignored by popular desktop environments such as Xfce, Gnome and KDE. It's apparent replacement notify-send gives the impression that it has never been tested in any other than the simplest cases, and that it is not ready for industrial strength use. Getting notify-send to work with NUT is not immediately evident, so although notify-send is not a part of NUT, we discuss this problem here.



Figure 105: Example of a notification.

D.1 What's wrong with notify-send?

The program notify-send is part of a set of programs which implement the Gnome Desktop Notifications Specification. The introduction says:

> ≪ This is a draft standard for a desktop notifications service, through which applications can generate passive popups to notify the user in an asynchronous manner of events. ... Example use cases include:
>
> - Scheduled alarm
> - Low disk space/**battery warnings** ... ≫

From this introduction it would seem that desktop notifications are exactly what is needed to present [OL]→[OB] and [OB]→[OB LB] warnings to the users, but unfortunately, things are not that simple.

Program notify-send is a utility which feeds message objects to a message server, such as `notifyd`. Taking the Xfce desktop environment as an example, Xfce provides it's message server called `xfce4-notifyd`. See `man xfce4-notifyd-config`, `man notify-send` and the Desktop Notifications Specification. There is also an `xfce4-notifyd` web page.

Experience shows that just calling notify-send in the script upssched-cmd does not work. The message simply disappears. Closer examination on the openSUSE distribution with command `ps -elf | grep ups` shows that if daemon upsmon running as user "upsd" calls notify-send to present a message, the notify daemon is launched with the same userid "upsd" as the caller. In Debian, NUT runs as user "nut" and the notify daemon is launched with the name userid "nut". *Users such as "upsd" and "nut" do not have access to the desktop environment.*

If the caller is the upsmon daemon which has no access to the desktop environment, then neither will the corresponding notification daemon. This is surprising. One would expect a design closer to that of the printer daemon `cupsd` which runs permanently in the background receiving files to be printed. There is only one daemon `cupsd` and that daemon isolates the user from needing to know how to drive printers.

To get the message to show on the user's screen appears to require two actions:

1. Give user "upsd" ("nut" on Debian) the right to act as any user,
2. Search for logged in users, and for each user construct the user's environment variable `DISPLAY`, and call utility notify-send as that user to notify the user.

D.2 Give user "nut" ("upsd") the right to act as any user

To improve security in NUT, the upsd and upsmon daemons is not executed as root, but rather as a non-root userid. This userid is typically called "nut" or "upsd". See table 104 for a list of possible users. We will use the name "nut". "nut" is not a regular user and does not have the access to the X-server needed to display data. This is a problem for the notification service, which we now fix.

Add the following lines to the file `/etc/sudoers`

```
749 # Host alias specification
750 Host_Alias LAN = 10.218.0/255.255.255.0,127.0.0.1,localhost,gold
751
752 nut  LAN = (ALL)  NOPASSWD:SETENV: /usr/bin/notify-send
```

Figure 106: Modifications to file `/etc/sudoers`

Line 750 corresponds to the editor's system and should be adapted to your setup.
On line 752 the directive `SETENV:` is needed for openSUSE but optional for Debian.
The file `/etc/sudoers` contains the following warning:

> *This file MUST be edited with the 'visudo' command as root. Failure to use 'visudo' may result in syntax or file permission errors that prevent sudo from running.*

See `man sudoers` and `man visudo`. The un-l33t do not have to use vi. Luckily, the command `VISUAL=/usr/bin/emacs visudo -f /etc/sudoers` also does the job.



D.3 Search for and notify logged in users

Figure 107 shows a Bash script notify-send-all which can be used in place of notify-send to send messages from upssched-cmd to all the X display users currently logged in. Script notify-send-all accepts as argument the message to be displayed. The message will be displayed indefinitely as "critical". The editor places the script in file `/usr/local/bin/notify-send-all`.

```
753 #! /bin/bash -u
754 # notify-send-all sends notifications to all X displays
755 # Assumes /etc/sudoers allows caller to sudo as any user.
756 # E.g. nut LAN = (ALL) NOPASSWD:SETENV: /usr/bin/notify-send
757 # Call with text to be displayed as argument.
758 XUSERS=( $( who | grep -E "\(:[0-9](\.[0-9])*\)" \
759          | awk '{print $1$NF}' | sort -u ) )
760 for XUSER in $XUSERS      # E.g. jschmo(:0)
761 do NAME=(${XUSER/\(/ })   # Insert space, make NAME an array
762    DISPLAY=${NAME[1]/)/}  # E.g. :0
763    sudo -u ${NAME[0]} DISPLAY=${DISPLAY} \
764             /usr/bin/notify-send -t 0 -u critical "$@"; RC=$?
765    if [[ $RC -ne 0 ]]; then exit $RC; fi
766 done
```

Figure 107: Bash script notify-send-all

Line 758 produces a Bash array of all the users identified by `who` who have X displays. Each item in the array corresponds to a logged in user with an X display and is of the form `jschmo(:0)`.

For each user logged in with an X display, line 761 creates a Bash array containing the user name and the X display number in the form `jschmo :0)`.

Line 762 extracts the X display number `:0` and on line 763 calls notify-send to notify the user as if user "nut" ("upsd" on openSUSE) was that logged in user. Note that environment variable `DISPLAY` is set for that user.

See the discussion "Show a notification across all running X displays" on the stackexchange site.

D.4 Testing the notify-send-all setup

A simple way of testing the use of notify-send if you are using the chapter 4 configuration is to simply disconnect the wall power for 10 seconds. This is sufficient to provoke upsmon into calling upssched-cmd which in turn calls notify-send-all as shown at line 194.

While wall power is disconnected, use a command such as `ps -elf | grep -E "ups[dms]|nut"` to find the programs running as user "nut" ("upsd" on openSUSE):

```
767 nut   2635     1  ... /usr/bin/usbhid-ups -a Eaton
768 nut   2637     1  ... /usr/bin/dummy-ups -a heartbeat
769 nut   2641     1  ... /usr/sbin/upsd
770 root  2645     1  ... /usr/sbin/upsmon
771 nut   2646  2645  ... /usr/sbin/upsmon
772 nut   3217     1  ... /usr/sbin/upssched UPS Eaton@localhost: On battery
773 nut   3236     1  ... dbus-launch --autolaunch=d1cd...ca5d2 ...
774 nut   3237     1  ... /bin/dbus-daemon --fork --print-pid 5 ...
775 nut   3241     1  ... /usr/lib/xfce4/notifyd/xfce4-notifyd
776 nut   3243     1  ... /usr/lib/xfce4/xfconf/xfconfd
```

Lines 767-772 are due to NUT activity, and lines 773-776 are due to the use of notify-send. Note on line 775 that the `xfce4-notifyd` daemon is running as user "nut "!

D.5 References for notify-send

1. For a suggestion of how to send notifications on an Apple Mac, see the posting by Robbie van der Walle, Sun Jun 11 11:27:55 UTC 2017, in the nut-upsuser mailing list.
2. For a discussion of how to send notifications to all running X-server users, see stackexchange question 2881.
3. The Gnome "Desktop Notifications Specification" is still a very long way from being RFC quality.
4. Man pages: See `man xfce4-notifyd-config` and `man notify-send`
5. Xfce4 web page: There is also an `xfce4-notifyd` web page.

These techniques have been tested with the Xfce desktop environment on openSUSE and Debian. The editor would be pleased to hear of any successful adoption of the techniques on Fedora, Arch or Ubuntu based systems, using other desktop environments such as Cinnamon, KDE or Gnome.



E Log rotation for upsdTLS.py and UPSmon.py

The well known Unix/GNU Linux utility program `logrotate` provides a convenient way of managing log files. See `man logrotate(8)`. NUT 2.7.4 already provides a declaration for it's log files. The following declaration provides separate management for the log files created by upsdTLS.py and UPSmon.py.

The file should be created as `/etc/logrotate.d/NUT` with ownership `root:root` and permissions `644`.

```
777 # Log rotation configuration for upsdTLS.py, UPSmon.py
778 # Rotate NUT log file either monthly or when exceeding 5 Mb
779 #
780 # For more information, refer to logrotate(8) manual page:
781 #   http://linuxcommand.org/man_pages/logrotate8.html
782 #
783 /var/log/NUT.log {
784     missingok
785     notifempty
786     size=5M
787     rotate 12
788     monthly
789     create 0600 nut nut
790 }
```

Figure 108: Log rotation for upsdTLS.py and UPSmon.py

Line 788 calls for a log rotation every month, and line 787 requires keeping 12 previous months' logs, so in all there will be one year's records.

Line 789 creates a file with owner `nut:nut` suitable for Debian. You should adapt this for your distribution. See table 104.



Part 4

UPS monitoring using Python3 script

Warning: This is Work in Progress

Part 1 of this documentation discussed the way in which UPS activity reported by upsd can be monitored using the monitoring software provided with NUT 2.8.0. This part 4 covers the use of Python3 scripts and openSSL/TLS to monitor the same UPS activity.

This Part provides descriptions of Python3 scripts UPSmon.py and mkUPSmonconf.py. The script UPSmon.py requires a helper script to create TLS certificates. The script mkNUTcert.py is described in part 2 chapter 10.

The scripts and their SHA1 check sums may be downloaded from http://rogerprice.org/NUT

F Python3 script UPSmon.py version 1.2

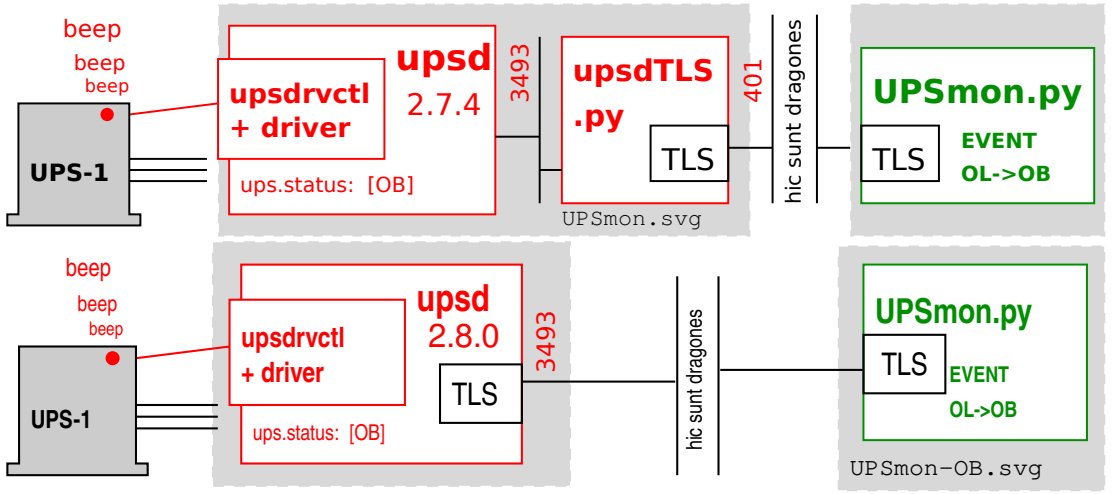


Figure 109: UPSmon.py requires TLS.

F.1 What is UPSmon.py ?

UPSmon.py is a Python3 script which replaces upsmon, upssched and upssched-cmd. The configuration files `upsmon.conf` and `upssched.conf` are replaced by a single configuration file `UPSmon.conf`. The current version 1.2 of UPSmon.py is "experimental", intended for experiment and demonstration.

F.1.1 Principal differences between upsmon and UPSmon.py

The principal differences between NUT's upsmon and UPSmon.py are:

1. UPSmon.py is written in Python3 rather than K&R C. It is hoped that this use of a well known higher level language will encourage further experimentation. The script is in one single file rather than the many separate files used in NUT C code. Like the NUT C code, the script is **not** object oriented. To assist further development, the script provides 116 error and warning messages, and the `-D` and `-Y` debug options provide a detailed "walk-through" of the script's operations.
2. Unlike upsmon, UPSmon.py does not retain the parent process when forking to a non-privileged user. This improves security, but implies that the non-privileged user such as `nut` has `sudo` rights for programs `wall`, `notify-send` and `shutdown`.
3. UPSmon.py assumes that it will be managing a large number of physical and virtual UPS and other power supply units. The management may be of the type "primary" or "secondary", known formerly as "master" or "slave", or simply as an observer with the primary/secondary shutdown decisions taken elsewhere.
4. The UPS units, real and virtual, are collected into groups. Every UPS must be in exactly one group. upsmon does not support groups.
5. All UPS's must be individually identified. Unlike NUT, there are no "wildcard" UPS's. Each UPS has a formal "fully qualified" name which is of the form *group*:*ups*@*host*:*port* , for example `HB:heartbeat@bigbox:3493` , although shortened forms are used where there is no ambiguity.
6. The configuration file `UPSmon.conf` is read by PLY, Python Lex and Yacc. This implies a slightly slower start-up than NUT but allows freer formats and many possibilities for future expansion.
7. The `upsmon.conf` declarations `DEADTIME`, `FINALDELAY`, `HOSTSYNC`, `NOCOMMWARNTIME` and `RBWARNTIME` are not needed in `UPSmon.conf` since they are timers which can be expressed directly if needed.
8. All communication between UPSmon.py and upsd is TLS encrypted. The version of OpenSSL used is too recent to be compatible with nut 2.7.4, so a shim front end for upsd called upsdTLS.py is provided to accept TLS encrypted commands from UPSmon.py and then relay that traffic to the local upsd. Part 2 describes upsdTLS.py. The options chosen for TLS call for the latest version with full checking of the certificates. Use of the earlier and now deprecated SSL is excluded.
9. UPSmon.py supports two loggers: the system log and a text based NUT-specific log.
10. UPSmon.py does not require a supplementary program such as upssched or a script such as upssched-cmd. The functions of those programs are available in UPSmon.py. NUT's upsmon

Action	Effect
`STARTTIMER` *name value*	Start timer with the given name and value in seconds.
`CANCELTIMER` *name*	Cancel timer with the given name.
`EMAIL` `FROM` *text* `TO` *text* `SUBJECT` *text* `MESSAGE` *text*	Send email.
`WALL` *text*	Send text to local wall.
`NOTIFY` *text*	Place text on screens of all logged-in local accounts.
`PRINT` *text*	Send text to STDOUT.
`EPRINT` *text*	Send text to STDERR.
`NUTLOG` *text*	Send text to NUT-specific logger.
`SYSLOG` *text*	Send text to system logger.
`SETFSD` *name*	Send FSD to upsd for UPS *name*.
`SHUTDOWN` *option when*	Shutdown the system, e.g. with `/usr/sbin/shutdown -h now` .
`DEBUG` *level*	Turn on/off the debugging output to the NUT log.

Figure 110: Actions provided by UPSmon.py.

provides three `NOTIFYFLAG` options: `SYSLOG`, `WALL` and `EXEC`, UPSmon.py replaces these with the more complete set of actions shown in figure 110.

11. Texts to be included in messages may be given names, and may incorporate other named messages. The upsmon `NOTIFYMSG %` substitution is extended to provide the substitutions shown in table 111.

`%(u)s`	Fully qualified name of the UPS unit
`%(c)s`	Current charge of the UPS unit
`%(e)s`	The event which has produced this message
`%(b)s`	A banner of the form “2020-08-15 upsd@bigbox”
`%(h)s`	The hostname, the name of the local machine

Figure 111: `%` substitutions available in messages.

12. The low battery status [LB] provided by upsd is supplemented by three further low battery statuses [LB1], [LB2] and [LB3] for which the trip levels may be set in `UPSmon.conf`.
13. When the sum of the `POWERVALUE` in a group with status [OL] does not meet the group's `MINSUPPLIES` requirement, UPSmon.py raises status [LS]. In upsmon this is implicit in the client's logic.

F.2 Compatibility with upsmon.

UPSmon.py can be run at the same time and in the same machine as upsmon. UPSmon.py does not interfere with direct access to upsd port 3493. Command line utility programs such as `upsc` still function normally.

F.3 Overview of UPSmon.py

The script has a configuration file, and many options. In general few options and in some simple cases none at all need be changed. To see the options and their default values you can enter command `UPSmon.py --help`

```
$ UPSmon.py --help
usage: UPSmon.py [-h] [--command fsd|reload|stop] [--config <file>]
       [--debug] [--debugYacc] [--logfile <file>] [--notify <executable>]
       [--PIDfile <file>] [--shell <executable>] [--sudo <executable>]
       [--testSHUTDOWNflag] [--upsdtimeout <float>] [--user <user>]
       [--version] [--wall <executable>]
```

Figure 112: Command `UPSmon.py --help`

.

Let's look at these optional arguments in more detail.

`-h, --help` Show this help message and exit.

`--command fsd|reload|stop` Send command to UPSmon.py process and exit. Valid commands are `fsd`, `reload`, `stop`.

`-c <file>, --config <file>` The configuration file. UPSmon.py tries to guess where you put this. Debian sysadmins might see `/etc/nut/UPSmon.conf` . OpenSUSE admins might see `/etc/ups/...` See table 104 for a list of possible directories.

`-D, --debug` Increase the debugging level, may be repeated but then you get more than any human can read. Debugging output is written into a NUT log file. This option does not cover Lex and Yacc.

`-Y, --debugYacc` Increase the debugging level for Lex and Yacc. No human being should ever be required to read this stuff. Debugging output is written into a NUT log file.

`-l <file>, --logfile <file>` The log file, with default `/var/log/NUT.log` . Progress and error messages and the stuff generated by options `-D` and `-Y` go into this file. Note that if upsdTLS.py and UPSmon.py are running in the same machine they will write into the same log. See chapter E for an extension to `logrotate` to cover this file.

`-n <executable>, --notify <executable>` The notification executable. The default is `/usr/bin/notify-send -t 0 -u critical`

`--PIDfile <file>` The child PID is written into this file, for the greater pleasure of systemd. The default is `/run/nut/UPSmon.pid` Do not change this unless you know what you are doing. You should also review the systemd service unit.

`--shell <file>` The shell that will process the `SHELLCMD` actions. The default is `/bin/bash -c`

`--sudo <executable>` Authorise user to execute code as another user. The default is `/usr/bin/sudo` . Use of `sudo` assumes that file `/etc/sudoers` allows the caller to sudo as the required user. For example

```
nut LAN = (ALL) NOPASSWD:SETENV: /usr/bin/notify-send, /usr/bin/wall
nut LAN = (ALL) NOPASSWD:SETENV: /usr/sbin/shutdown
```

where LAN is defined by a declaration such as

```
Host_Alias LAN = 10.218.0/255.255.255.0, 127.0.0.1, localhost
```

To update `/etc/sudoers` , use `visudo` , for example `VISUAL=/usr/bin/emacs visudo -f /etc/sudoers` .

`-K <executable>, --testSHUTDOWNflag` Test the `SHUTDOWN` flag. *Not implemented.*

`--upsdtimeout <float>` Socket timeout for exchanges with upsd. The default is 5.0 seconds.

`-u <user>, --user <user>` After launch as root, run as this user. UPSmon.py tries to guess the user. OpenSUSE admins would probably see `upsd`, whereas Debian admins would see `nut`. See table 104 for a list of possible users.

`-v, --version` Show program, Python and SSL/TLS versions, then exit.

`-w <executable>, --wall <executable>` The wall executable. The default is `/usr/bin/wall`

F.4 Running UPSmon.py

It is possible, in a simple installation, to run the daemon UPSmon.py in the same machine as upsd. However the design is for remote monitoring of one or more upsd servers across a hostile network. UPSmon.py assumes that the server(s) is/are already running and ready to receive the `STARTTLS` command.

If you use systemd to manage your box, then you will need to create a new service unit, since systemd is unable to start two forking services from the same unit. See `man systemd.service(5)`. There can only be one `Type=forking` per unit.

Copy the file `/usr/lib/systemd/system/nut-monitor.service` to `/etc/systemd/system/nut-py-monitor.service` and modify the new file shown in figure 113. Lines 799, 801 and 802 have been changed.

```
797 [Unit]
798 Description=Network UPS Tools - Python - power device monitor
799 After=local-fs.target network.target

800 [Service]
801 ExecStart=/usr/sbin/UPSmon.py
802 PIDfile=/run/nut/UPSmon.pid
803 Type=forking

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

Figure 113: systemd service unit `nut-py-monitor.service` for UPSmon.py.

You may choose to place the UPSmon.py script in directory `/usr/sbin/` or make `/usr/sbin/UPSmon.py` a link to wherever you put the Python script. Note that systemd service units in `/etc/` take precedence over those in `/usr/lib/`. See `man systemd.unit(5)`. After you have made the changes, you should run the command `systemctl daemon-reload` . See `man systemctl(1)`. Before running upsdTLS.py the first time, you will need to run the command

```
systemctl enable nut-py-monitor.service
```

The following `systemctl` commands will be of use to you:

`systemctl daemon-reload` to make any changes to the service unit available to systemd.

`systemctl enable nut-py-monitor.service` to make the daemon UPSmon.py operational and "startable".

`systemctl start nut-py-monitor.service` to start UPSmon.py. Note that this will not erase the log file. If you want to clear the log file then you need to do that yourself. See also chapter E for a discussion of log rotation.

`systemctl status nut-py-monitor.service` to see the current status of daemon UPSmon.py.

`systemctl stop nut-py-monitor.service` to stop UPSmon.py.

UPSmon.py should start automatically when the system starts, but it can also be stopped and started manually with the `systemctl` commands.

Serious errors will prevent UPSmon.py from starting and you can read about them in the NUT log and in the system log. After starting UPSmon.py, check the NUT log for warnings and other error messages. Look for the reports beginning "`Sanity checks for this configuration ...`".



F.5 UPSmon.py's events based un upsd's status changes

Status change symbol		Event based on Status Change
None->ALARM	ALARM->None	The UPS has raised/dropped the alarm signal.
None->BOOST	BOOST->None	The UPS is now boosting/not boosting the output voltage.
None->BYPASS	BYPASS->None	The UPS is/is not now bypassing its own batteries.
None->CAL	CAL->None	The UPS is/is not now in calibration mode.
None->CHRG	CHRG->None	The UPS is/is not now recharging its batteries.
None->DISCHRG	DISCHRG->None	The UPS is/is not now discharging its batteries.
None->LB	LB->None	The driver says the UPS battery charge is now low/no longer low with respect to level defined by upsrw. See chapter 2.10.
None->OFF	OFF->None	The driver says the UPS is/is not now OFF.
OL->OB	OB->OL	The UPS is now on battery/no longer on battery.
None->OVER	OVER->None	The UPS is/is not now in status [OVER].
None->RB	RB->None	The UPS needs/no longer needs to have its battery replaced.
None->TEST	TEST->None	The UPS is/is not now performing a test.
None->TRIM	TRIM->None	The UPS is now trimming/not trimming the output voltage.

Figure 114: Symbols used to represent events monitored by UPSmon.py.

UPSmon.py, like NUT's upsmon is an example of a client of upsd[33]. Just as upsmon does, it runs permanently as a daemon in a local or remote box, polling the status changes of the UPS unit. It is able to react to changes in the UPS state for example by emitting warning messages, or shutting down the box. The actions are specified in the configuration file `UPSmon.conf` which will be discussed in specific examples.

As the state of a UPS evolves, each status change, called an "EVENT", is identified with the symbols shown in figure 114. *(These correspond to the NOTIFY events, also known as a "notifytype" in NUT.)*

For example, figure 109 shows what happens when wall power fails. Daemon upsd has polled the UPS, and has discovered that the UPS is supplying power from it's battery. The `ups.status` changes to [OB]. Daemon UPSmon.py has polled upsd, has discovered the status change and has generated the OL->OB event.

[33]See chapter 1.3 for details of upsd.

F.5.1 UPSmon.py's additional status symbols and events

Other statuses generated by UPSmon.py	
[COMM]	UPSmon.py, via upsd is in effective communication the the UPS.
[NOCOMM]	UPSmon.py is no longer in effective communication the the UPS.
[FSD]	The UPS unit is in Forced ShutDown mode.
[LB1]	The battery charge is below a critical level specified by declaration `LET battery.charge.low.1 = 'L'`.
[LB2]	The battery charge is below a critical level specified by declaration `LET battery.charge.low.2 = 'L'`.
[LB3]	The battery charge is below a critical level specified by declaration `LET battery.charge.low.3 = 'L'`.
[LS]	Within a group, the total power value of the UPS units with status [OL] does not satisfy the group's MINSUPPLIES declaration.
[TICK]	Status generated by a heartbeat UPS.
[TOCK]	Status generated by a heartbeat UPS.
[TO]	A UPSmon.py timer has completed.

Figure 115: Additional status symbols generated by UPSmon.py.

Status change symbol		Other Events generated by UPSmon.py
COMM->NOCOMM	NOCOMM->COMM	Communication with the UPS in now lost/restored.
None->FSD	FSD->None	The UPS is/is not now in Forced ShutDown mode.
None->LB1	LB1->None	The UPS battery charge is now low/no longer low with respect to critical level 1.
None->LB2	LB2->None	The UPS battery charge is now low/no longer low with respect to critical level 2.
None->LB3	LB3->None	The UPS battery charge is now low/no longer low with respect to critical level 3.
None->LS	LS->None	Within a group, the total power value of the UPS units with status [OL] does/does not satisfy the MINSUPPLIES declaration.
None->TICK	TICK->None	A heartbeat UPS has/has not generated a [TICK].
None->TOCK	TOCK->None	A heartbeat UPS has/has not generated a [TOCK].
TO->my-timer		Timer "my-timer" has completed.

Figure 116: Additional events monitored by UPSmon.py.

In addition to the events based on upsd status changes, UPSmon.py also generates further statuses and status changes based on its monitoring of upsd. See figure 115. Changes in these additional statuses give rise to additional events shown in figure 116.

F.6 Configuration file

There is just one configuration file for UPSmon.py which replaces `upsmon.conf`, `upssched.conf` and upssched-cmd. The formal grammar for this configuration file is in chapter H. The file contains:

1. Comments and blank lines. A comment begins with a `#` character found outside a quoted text, and continues up the the end-of-line.
2. Initial declarations. See section F.6.1
3. One or more group declarations. See section F.6.2.

The following technical terms are used in the descriptions of the configuration file:

quotation mark One of the following five styles[34] of text marker. See chapter G for help in typing the characters which may not be on your keyboard.

1. double quotation marks: "*bla..bla...*" which are probably on your keyboard,
2. single quotation marks: '*bla..bla...*' which are also on your keyboard,
3. french guillemets: «*bla..bla...*»,
4. mathematical left ceiling/right floor ⌈*bla..bla...*⌋ and
5. corner brackets used for quotations in asian lanuages: ⌜*bla...bla...*⌟.

quotetext A text in quotation marks. E.g. «Hello World»

quotetexts A sequence of one or more *quotetext* declarations. E.g. «Today is » «Friday.» This results in a single text "Today is Friday."

number An integer or floating point number such as 15 or 2.8.

name Names for groups, timers, UPS's, messages. The name begins with `[a-zA-Z_]` and continues with as many of `[a-zA-Z0-9._%+-]` as you like. E.g. `UPS31.a-BIG_BOX`.

ups-name All UPS's must be individually identified. Unlike NUT, there are no "wildcard" UPS's. Each UPS has a formal "fully qualified" name which is of the form *group*:*ups*`@`*host*:*port* for example `HB:heartbeat@bigbox:3493` , although shortened forms are used where there is no ambiguity.

[34] I couldn't decide which ones to use so I kept them all. Ed.

F.6.1 Initial declarations

The initial declarations are

`SMTPSERVER` *quotetext* `PORT` *number* `USER` *quotetext* `PASSWORD` *quotetext* If you want to send e-mails, you must provide details of your e-mail service provider. For example `SMTPSERVER 'mail.gandi.net' PORT 465 USER 'mbox@example.com' PASSWORD «1234»`. Connections with the SMTP server are always TLS encrypted.

`LET` *name* `=` *quotetexts* Provide a name for one or more *quotetext*. This saves a lot a typing. For example `LET banner = ⌜[%(b)s] UPS=%(u)s charge=%(c)s event=%(e)s⌟`. The named message `LET hostname =` *hostname* is built in. There may be multiple `LET` declarations, and each may make use of names declared in previous `LET`s.

`MAXNOTIFY` *number* This limits the number of on-screen notifications, and was needed during early debugging when things often exploded. It will probably be removed in the future. The default is 20.

`POLLFREQ` *number* This is the polling period for all UPS units managed by this UPSmon.py instance. The default, which is the recommended value, is 5 seconds. See also `man upsmon.conf`

`POLLFREQALERT` *number* This is the polling period for all UPS units managed by this UPSmon.py instance when any one of them is in status [OB]. The default is 5 seconds.

F.6.2 Group declarations

Each group in a sequence of groups begins with a `GROUP` declaration header followed by other declarations described in this section.

Within a group, a *condition* is either empty or has the form `IF` *old-status* `->` *new-status*. The condition has the value True if in the sequence of events from the given UPS, that UPS now has status *new-status*. For example the expression `IF OB -> OL` is True if the UPS currently has status [OL] and False if the UPS has status [OB]. Note that *old-status* -> *new-status* must be a valid event as listed in chapter F.5.

The `GROUP` declaration header is as follows:

`GROUP` *name* `HOST` *name* `PORT` *number* `CERTFILE` *name/quotetext* One or more UPS units share the same `HOST`, `PORT` and TLS `CERTFILE`. E.g. `GROUP LOCAL HOST localhost PORT 401 CERTFILE /etc/nut/gold-client.cert.pem` . The UPS units attached to this host are grouped together and each is specified by a `MONITOR` declaration in this group.

Within each group the following declarations may appear:

`LET` *name* `=` *quotetexts* Further named texts. Note that there is only one name space shared by all `LET` declarations. It's up to you to avoid clashes.

The *name* `battery.charge.low.`*i* for $i = 1..3$ is a special case in which the *quotetexts* must be quoted integer. The effect is to assign the integer value as the battery charge level at which the events None->LB*i* and LB*i*->None will occur. For example `LET battery.charge.low.2 = '33'` The level is set for the most recently defined UPS, i.e. the previous `MONITOR` declaration. The default levels are LB1=50, LB2=25 and LB3=12.

`MONITOR` *ups-name* `POWERVAL` *number* `UPSDUSER` *name* `PASSWORD` *quotetext* `TYPE` *name*
Each UPS unit to be managed must be declared. The *ups-name* must match the name in the `ups.conf` declaration. See for example line 32. The `POWERVAL` is the number of power supplies that this UPS feeds. The `UPSDUSER` is the "user" declared in `upsd.users`. See line 40. The `PASSWORD` is the value declared in `upsd.users`. See line 41. The `TYPE` value must be `primary` or `secondary`. The earlier values `master`, `slave` are accepted. In NUT's `upsmon.conf` `primary` means this system will shutdown last, allowing any secondaries time to shutdown first. The declaration is included here to facilitate interworking with upsmon but in UPSmon.py, it is merely a declaration of intention, since the logic is decided by the declared actions.

E.g. `MONITOR ups1 POWERVAL 1 UPSDUSER leboss PASSWORD 'sekret' TYPE primary`

`MINSUPPLIES` *number*

Declare for each `GROUP` the number of power supplies which must be operational, and that if fewer are available, NUT must shut down the server. The default value is 1 if this declaration is omitted. See chapter 3.2

More work needed here to create a MINSUPPLIES event.

`WHEN` *ups-name* `REPORTS` *old-status->new-status* `:` *actions*
Declare what, if anything, is to be done when an event, i.e. a status change occurs. The *ups-name* may be abbreviated when there is no ambiguity, but the fully qualified UPS name is always used internally.

The sequence *old-status->new-status* defines a status change, i.e. an event. The valid events are listed in chapter F.5.

When the event specified for this UPS is detected, the *actions* will be executed. For example `WHEN ups1 REPORTS None->LB :` *actions* Let's hope those actions do something useful.

`WHEN` *ups-name* `TIMEOUT` *timer-name* `:` *actions*
Declare what, if anything, is to be done when a timeout occurs. The *timer-name* will have been declared by a previous `STARTTIMER` action. `TIMEOUT` may be written as `TO`. For example `WHEN ups1 TO final-delay : SHUTDOWNCMD «/sbin/shutdown -h now»`

actions A sequence of one or more of the following:

condition CANCELTIMER *timer-name* The *timer-name* must have been declared by a previously[35] executed STARTTIMER action. It is not an error to cancel a timer after it has run out.

condition DEBUG 0/1/2 Initiate or terminate debugging output. Note that since a set of actions is executed in random order, you should not rely on a DEBUG in the same set of actions as the action you wish to trace.

condition EMAIL FROM *quotetext*
TO *quotetext*
SUBJECT *quotetext*
MESSAGE *quotetexts*

Send an email via the mail server declared in the introduction by SMTPSERVER. E.g.

```
EMAIL FROM «UPSmon.py@example.com»
      TO   «sysadmin@bigbox.com»
      SUBJECT «Msg-1-min»
      MESSAGE «Msg-1-min»
```

Where Msg-1-min has been previously declared in a LET. Note that the message must be in 7-bit ascii. Any character more exotic will be converted to a "~".

condition STARTTIMER *timer-name number* Declare and start a timer with the given name, and the given value in seconds. It is up to you to avoid name conflicts between timers and with other names. E.g. STARTTIMER final-delay 5

condition EPRINT *quotetexts* Send the *quotetexts* to STDERR. When UPSmon is daemonized, EPRINT is ignored. Use NUTLOG instead.

condition NOTIFY *quotetexts* Place the *quotetexts* in an on-screen notification for all logged-in users. If UPSmon.py is run as a non-privileged user, which is usually the case, than that user, for example nut, must be given access to program notify-send in file /etc/sudoers . See chapter D.2 for details of how to do this. See also man sudo(8) for lots and lots of brain-damaging detail.

condition NUTLOG *quotetexts* Write the *quotetexts* into the NUT log file specified by option --logfile. The *quotetexts* will be prepended with a timestamp and a reminder of the source program and line number. For example action NUTLOG «Hello World» might add the following line to the log file:

```
18:32:25 UPSmon.py[3498] Hello World
```

See chapter E for an extension to logrotate to cover this file.

condition PRINT *quotetexts* Send the *quotetexts* to STDOUT. When UPSmon is daemonized, PRINT is ignored. Use NUTLOG instead.

[35] "Previous" means previous in time, not in the order of declarations in UPSmon.conf.

condition `SETFSD` *ups-name* This action sets the "forced shutdown" flag on each secondary (slave) UPS when the primary (master) plans to power it off. This is done so that secondary (slave) systems will know about the power loss and shut down before the UPS power disappears. UPSmon.py, like upsmon, in primary (master) mode is the primary user of this function.

Setting this flag makes [FSD] appear for this UPS. This [FSD] should be treated just like a [OB LB]. To use this action, you need `upsmon primary` in `upsd.users`, or "FSD" action granted in `upsd.users`. See `man upsd.users`.

Note that [FSD] in upsd is currently a latch - once set, there is no way to clear it short of restarting upsd. This may cause issues when upsd is running on a system that is not shut down due to the UPS event.

See the Network UPS Tools Developer Guide, Network protocol information

condition `SHELLCMD` *quotetexts* Call on the shell defined by the option `--shell` to execute the command given by the *quotetexts*. For example

```
SHELLCMD «echo "Today is $(date)" >> /var/log/NUT.log»
```

might write "`Today is Tue Oct 13 10:09:02 CEST 2020`" into the log file.

condition `SHUTDOWNCMD` *quotetexts* Call for a system shutdown using the command specified by the *quotetexts*. For example, `SHUTDOWNCMD «/sbin/shutdown -h 0»`. If UPSmon.py is run as a non-privileged user, which is usually the case, than that user, for example `nut`, must be given access to program `shutdown` in file `/etc/sudoers` . See chapter D.2 for details of how to do this. See also `man sudo(8)` for lots of detail.

condition `SYSLOG` *quotetexts* Write the *quotetexts* into the system log. The system log provides 8 levels of urgency. They are shown, in order of decreasing importance, in table 117. If your *quotetexts* are prefixed with one of these urgency indicators, your message will be logged at the required level e.g. `SYSLOG «[debug]» « UPS %(u)s burning»` . The default level is `[info]`.

`[emerg]`	System is unusable
`[alert]`	Action must be taken immediately
`[crit]`	Critical conditions
`[err]`	Error conditions
`[warning]`	Warning conditions
`[notice]`	Normal, but significant, condition
`[info]`	Informational message (default)
`[debug]`	Debug-level message

Figure 117: System log urgency levels.

condition `WALL` *quotetexts* Place the *quotetexts* in a console message for all logged-in users. If UPSmon.py is run as a non-privileged user, which is usually the case, than that user, for example `nut`, must be given access to program `wall` in file `/etc/sudoers` . See chapter D.2 for details of how to do this. See also `man sudo(8)` for details. Note that `wall` does not support UTF-8.

G Typing alternative text bracketing characters

Text in `UPSmon.conf` must be in brackets. You are free to choose which style; the following table may help you to type styles which are not on your keyboard.

Unicode		Emacs	Vim	Full name
"	U+0022	Keyboard "	Keyboard "	QUOTATION MARK (Used left and right)
’	U+0027	Keyboard ’	Keyboard ’	APOSTROPHE (Used left and right)
«	U+00AB	`AltGr{` or `Ctl-q 00ab`	`AltGr{` or `Ctl-v u00ab`	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
»	U+00BB	`AltGr}` or `Ctl-q 00bb`	`AltGr}` or `Ctl-v u00bb`	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
⌈	U+23A1	`Ctl-q 23a1`	`Ctl-v u23a1`	LEFT SQUARE BRACKET UPPER CORNER
⌋	U+23A6	`Ctl-q 23a6`	`Ctl-v u23a6`	RIGHT SQUARE BRACKET LOWER CORNER
⌜	U+2E22	`Ctl-q 2e22`	`Ctl-v u2e22`	TOP LEFT HALF BRACKET
⌟	U+2E25	`Ctl-q 2e25`	`Ctl-v u2e25`	BOTTOM RIGHT HALF BRACKET

Figure 118: Alternative text bracketing characters.



H Grammar for UPSmon.conf

The UPSmon.conf file is parsed using David Beazley's PLY[36]. This is a pure Python approach to Lex and Yacc. There are no separate Lex and Yacc files. For background reading see "*lex & yacc*" by John R. Irvine, Tony Mason and Doug Brown, O'Reilly, first published 1990, ISBN: 1-56592-000-7.

The PLY's Lex and Yacc produce an abstract syntax tree known as AST. This is then interpreted as instructions to create a new configuration. If there are no errors, the new configuration is passed to UPSmon.py, otherwise UPSmon.py continues with the previous configuration. You can see AST in the log file if you run UPSmon.py with option -D.

H.1 Lexical structure

The configuration file is assumed to be encoded in UTF-8, and contains comments, tokens (keywords and symbols), numbers and quoted text interspersed with white space.

Whitespace Whitespace is any combination of the characters space, tab and newline. Whitespace serves only to separate the other components of a configuration file.

Comments The character # outside a quoted text begins a comment which continues up to the end of the line. The comment is ignored by the parser. A # inside a quoted text does not begin a comment. This is the same comment style as upsmon.conf and many other configuration files.

Names Names are labels which identify UPS units, timers, named messages, ... They are not quoted and are made up of the 67 characters a-zA-Z0-9._%+- . The leading character must be one of the 53 characters a-zA-z_ .

Numbers Numbers are non-negative and may be floating point. They are not quoted. E.g. 5.5 .

Tokens The tokens are names given to every piece of input that is recognisable by the lexer. They are shown in figure 120. The tokens are presented in the order in which they are tested by the lexer.

Quoted text Text is always quoted. The possible quotation marks are shown in figure 118. E.g. "text", 'text', «text», ⌈text⌋ and ⌜text⌟. A quoted text may not contain a newline or it's terminating quote character. E.g. «te»xt» is an error as is «te
xt».

Statuses The lexer recognises the following UPS statuses: None ALARM BOOST BYPASS CAL CHRG DEAD DISCHRG FSD LB COMM OB OFF OL OVER RB TEST TICK TOCK TRIM

Events An event is a transition from one status to another, and is seen by the lexer as STATUS RARR STATUS, e.g. None->LB .

[36] See David Beazley's PLC (Python Lex-Yacc) page at https://www.dabeaz.com/ply/

Token	Use
APCUPSDUSER	Keyword
CANCELTIMER	Keyword
CERTFILE	Keyword
COLON	Symbol :
DEBUG	Keyword
EMAIL	Keyword
EPRINT	Keyword
EQ	Symbol =
FROM	Keyword
GROUP	Keyword
HOST	Keyword
IF	Keyword
LET	Keyword
MAXNOTIFY	Keyword
MESSAGE	Keyword
MINSUPPLIES	Keyword
MONITOR	Keyword
NAME	Start with a-zA-z_ then a-zA-Z0-9._%+-
NOTIFY	Keyword
NUMBER	0 through 9 plus .
NUTLOG	Keyword
PASSWORD	Keyword
POLLFREQALERT	Keyword
POLLFREQ	Keyword
PORT	Keyword
POWERVAL	Keyword
PRINT	Keyword
QUOTETEXT1	'text'
QUOTETEXT2	"text"
QUOTETEXT3	«text»
QUOTETEXT4	⌈text⌋
QUOTETEXT5	⌜text⌟

Figure 119: UPSmon.conf lexer tokens.

Token	Use
RARR	Symbol ->
REPORTS	Keyword
SETFSD	Keyword
SHELLCMD	Keyword
SHUTDOWNCMD	Keyword
SMTPSERVER	Keyword
STARTTIMER	Keyword
STATUS	See status list
SUBJECT	Keyword
SYSLOG	Keyword
TIMEOUT	Keyword
TO	Keyword
TYPE	Keyword
UPSDUSER	Keyword
USER	Keyword
WALL	Keyword
WHEN	Keyword
ignore	Ignore spaces and tabs
ignore_COMMENT	Ignore #...
newline	Line counter

Figure 120: UPSmon.conf lexer tokens.



H.2 Yacc Grammar

The grammar shows the logical structure of the configuration file. There is no separate "yacc" grammar file. The productions are represented by functions such as the one shown in figure 121.

```
806 def p_configuration (p) :
807   'configuration : intros groups'
808   tag = ('configuration', p.lineno(len(p)-1)//LN, p.lineno(len(p)-1)%LN)
809   AST = (tag, p[1], p[2])
```

Figure 121: Representation of grammar production

Line 806 declares the function providing the grammar production seen in line 807 for the `configuration` production. The result is tagged with a 3-tuple seen in line 808 giving the identity, line number and column number, and forms the basis for the abstract syntax tree `AST`. The values for `p[1]` and `p[2]` in line 809 are provided by functions `p_intros` and `p_groups`.

Production	Notes
`configuration : intros groups`	Start here
`intros : intro` `| intros intro`	Start of introduction

`intro : smtp`
`| let`
`| pollfreqalert`
`| pollfreq`	

`smtp : SMTPSERVER quotetext PORT number` `USER quotetext PASSWORD quotetext`	
`let : LET name EQ quotetexts`	`battery.charge.low.`i for $i = 1..3$ the `name` is a special value.
`number : NUMBER`	
`pollfreqalert : POLLFREQALERT number`	
`pollfreq : POLLFREQ number`	End of the introduction

continued ...

Figure 122: `UPSmon.conf` grammar.

... continued

groups	: group | groups group_element	Start of group specs
group_element	: group_name | group_host | group_port | certfile | let | monitors | minsupplies | action_declarations	
group_name	: GROUP name	
name	: NAME	
group_host	: HOST name	
group_port	: PORT number	
certfile	: CERTFILE quotetext | CERTFILE name	
monitors	: monitor | monitors monitor	
monitor	: MONITOR name POWERVAL number user PASSWORD quotetext TYPE name	
user	: UPSDUSER name | APCUPSDUSER name	
minsupplies	: MINSUPPLIES number	
action_declarations	: action_declaration | action_declarations action_declaration	
action_declaration	: event_key actions	
event_key	: WHEN name TO name COLON | WHEN name TIMEOUT name COLON | WHEN name REPORTS STATUS RARR STATUS COLON	TO ≡ TIMEOUT
actions	: action_element | actions action_element	

continued ...

Figure 123: UPSmon.conf grammar, continued.

... continued	
action_element : condition cancel_timer | condition debug_level | condition email | condition start_timer | condition EPRINT quotetexts | condition NOTIFY quotetexts | condition NUTLOG quotetexts | condition PRINT quotetexts | condition SETFSD name | condition SHELLCMD quotetexts | condition SHUTDOWNCMD quotetexts | condition SYSLOG quotetexts | condition WALL quotetexts	
condition : IF STATUS RARR STATUS | empty	
quotetexts : quotetext | name | quotetexts quotetext | quotetexts name	
quotetext : QUOTETEXT1 | QUOTETEXT2 | QUOTETEXT3 | QUOTETEXT4 | QUOTETEXT5	
cancel_timer : CANCELTIMER name	
debug_level : DEBUG number	0, 1 or 2
start_timer : STARTTIMER name number	
email : EMAIL from to subject content	
from : FROM quotetext	
to : TO quotetext	
subject : SUBJECT quotetext	
content : MESSAGE quotetexts	
empty :	

Figure 124: UPSmon.conf grammar, final part.

I UPSmon.py configuration

A configuration file `UPSmon.conf` must be created to tell UPSmon.py how to handle the status changes coming from upsd. As with `upsmon.conf`, this can be done manually, but for simple cases, probably the majority, in which upsd and UPSmon.py run in the same machine, UPSmon.py provides a Python3 tool mkUPSmonconf.py, to create a complete fully functioning configuration file. You can either use the output of this tool or take it as the starting point for a customised configuration.

I.1 Configuration tool mkUPSmonconf.py version 1.3

```
$ mkUPSmonconf.py --help
usage: mkUPSmonconf.py [-h] [--configurationfile <filename>]
  [--clientcertfile <filename>] [--emailfrom <string>] [--emailto <string>]
  [--plan standard|timed] [--smtppass <string>] [--smtpport <integer>]
  [--smtpserver <domain>] [--smtpuser <name>] [--ups <name>]
  [--upsdname <name>] [--upsdpass <string>] [--upsdport <integer>]
  [--upsduser <name>] [--version]
```

Figure 125: Command `mkUPSmonconf.py --help`

.

mkUPSmonconf.py is a Python3 script which will build a simple configuration file `UPSmon.conf` for UPSmon.py. The script proposes a possible configuration file based on it's built-in default values and asks for your approval. Here is a fanciful example.

```
$ mkUPSmonconf.py
Here are your chosen values:
   --configurationfile /etc/nut/UPSmon.conf
   --clientcertfile /etc/nut/titan-client.cert.pem
   --emailfrom "<bigserver@BigU.edu>"
   --emailto "Big Joe <jschmo@BigU.edu>"
   --plan timed
   --smtpserver "mailbox@mailserver,com"
   --smtpport 465
   --smtpuser jschmo
   --smtppass qwertyuiop
   --ups UPS-123
   --upsdname localhost
   --upsdport 401
   --upsduser nut-admin
   --upsdpass sekret
If this configuration is correct, enter yes to proceed, anything else to exit:
```

If the proposed configuration values are satisfactory, you reply `yes`, but if they are not right for you, you hit Enter and rerun the script using the options to specify your preferred values.

The status is "experimental". The script is intended for demonstration and experiment. The license is GPL v3 or later at your choice, with support in the `nut-upsuser` mailing list.

Let's look at these arguments in more detail.

`-h, --help` Show this help message and exit.

`--configurationfile <filename>` The file which holds UPSmon.py's configuration. E.g. A Debian sysadmin might use `/etc/nut/UPSmon.conf`

`--clientcertfile <filename>` The file which holds the client's public TLS certificate required to access the server upsd, possibly with upsdTLS.py. E.g. A Debian sysadmin might use `/etc/nut/bigbox-client.cert.pem`

`--emailfrom <string>` The email address from which messages will be sent.
E.g. `"<bigserver@bigU.edu>"` Note the email convention of placing the address in angle brackets, and the double quotes needed to prevent Bash from interpreting the angle brackets.

`--emailto <string>` The email address of the person to whom messages will be sent.
E.g. `"Big Joe <jschmoe@bigU.edu>"` Note the email convention of placing the address in angle brackets, and the double quotes needed to prevent Bash from interpreting the angle brackets.

`--plan standard|timed` Specify standard or timed shutdown plan. Valid options are `standard` or `timed`.

`--smtppass <string>` The password for your account on the e-mail server. E.g. `qwertyuiop` . This definitely needs changing.

`--smtpport <integer>` Your e-mail server's TLS port. E.g. `465` . Communication with the mail server is always TLS encrypted.

`--smtpserver <domain>` Your e-mail server.
E.g. `mailbox.mailserver.com`

`--smtpuser <name>` Your sign-in account name on the e-mail server.
E.g. `mailbox@mydomain.com`

`--ups <name>` The name of your UPS, for example `UPS_123`. If you have more than one UPS unit then create a configuration file for the first, and then extend it using copy/paste of the actions for the second.

`--upsdname <name>` The name of the system on which upsd runs. E.g. `localhost` if UPSmon.py and upsd run on the same machine.

`--upsdpass <string>` The password for this upsd user, as given in `upsd.users`. Do you remember the `password = sekret` on line 41?

`--upsdport <integer>` The TLS port used by upsd possibly with shim upsdTLS.py. E.g. `401`

`--upsduser <name>` User for this UPS, as given in `upsd.users`. E.g. `nut-admin` on line 40

`-v, --version` Show program and Python versions, then exit.

I.2 `UPSmon.conf` configuration examples

Let's look at a shutdown plan generated by mkUPSmonconf.py.

I.2.1 Timed shutdown plan, part 1 of 4, the introduction

```
834 # UPSmon.conf timed shutdown plan generated by mkUPSmonconf.py version 1.3
                on 2022-09-03 17:18:48.202107
835 # Python version 3.9.2 (default, Feb 28 2021, 17:03:44)  [GCC 10.2.1 20210110]
836 # Calling command:
    ./mkUPSmonconf.py --plan timed --ups UPS-1 --upsdname localhost --upsdport 401
      --clientcertfile /etc/nut/titan-client.cert.pem --upsduser nut-admin
      --upsdpass sekret --smtpserver mail.gandi.net --smtpport 465 --smtppass qwerty
      --smtpuser mailbox@rogerprice.org --configurationfile /etc/nut/UPSmon.conf
      --emailfrom "<UPSmon@rogerprice.org>" --emailto "Price <roger@rogerprice.org>"
837 # Documentation: http://rogerprice.org/NUT/ConfigExamples.A5.pdf
838 # Support: nut-upsuser mailing list.

839 # All groups share the same POLLFREQ and POLLFREQALERT and e-mail relay
840 POLLFREQ 5.0  POLLFREQALERT 5.0
841 SMTPSERVER «mail.gandi.net» PORT 465
842      USER «mailbox@rogerprice.org» PASSWORD «qwertyuiop»

843 # Named messages  Let hostname = hostname is built in.
844 LET banner        = ⌜[%(b)s] UPS=%(u)s charge=%(c)s event=%(e)s␣
845 LET Msg-COMM      = banner " I have re-established communication with this UPS."
846 LET Msg-NOCOMM    = banner " I have lost communication with this UPS."
847 LET Msg-OL        = banner " Power restored, shutdown cancelled."
848 LET Msg-RB        = banner " Battery needs replacement."
849 LET Msg-shutdown  = banner " On battery, shutting down now ..."
850 LET Certfile      = «/etc/nut/titan-client.cert.pem»
```

Figure 126: Timed shutdown plan, part 1 of 4, the introduction.

Notes on figure 126

1. The command used to generate the file is repeated on line 836.
2. The POLLFREQ and POLLFREQALERT on line 840 are the same as upsmon. See chapter 4.1.
3. On line 841 the PORT number corresponds to a TLS port. Communication with the email service provider is always TLS encrypted.
4. On lines 841-842 the «...» is added automatically by the mkUPSmonconf.py script. You do not have to do this.
5. Line 850 corresponds to a Debian installation. See table 104 for a list of possible directories.

I.2.2 Timed shutdown plan, part 2 of 4, the shutdown

```
851 # The local UPS units
852 GROUP LOCAL HOST localhost PORT 401 CERTFILE Certfile
853 MONITOR UPS-1 POWERVAL 1 UPSDUSER nut-admin PASSWORD «sekret» TYPE primary

854 # Timed plan additional messages and actions
855 LET Msg-2-min = banner " On battery, shutdown in 2 mins, save your work ..."
856 LET Msg-1-min = banner " On battery, shutdown in 1 min, save your work ..."
857 WHEN UPS-1 REPORTS OL->OB :   NOTIFY Msg-2-min NUTLOG Msg-2-min
858                               STARTTIMER two-min 120 STARTTIMER one-min 60
859 WHEN UPS-1 TIMEOUT one-min :  NOTIFY Msg-1-min NUTLOG Msg-1-min WALL Msg-1-min
860                               EMAIL FROM « <UPSmon@rogerprice.org> »
861                                     TO   « Roger Price <roger@rogerprice.org> »
862                                     SUBJECT «Msg-1-min»
863                                     MESSAGE «Msg-1-min»
864 WHEN UPS-1 TIMEOUT two-min :  NOTIFY Msg-shutdown NUTLOG Msg-shutdown
865                               WALL Msg-shutdown STARTTIMER final-delay 5
866 WHEN UPS-1 REPORTS OB->OL :   NOTIFY Msg-OL NUTLOG Msg-OL WALL Msg-OL
867                               CANCELTIMER two-min CANCELTIMER one-min
                                  CANCELTIMER final-delay
868 # End of timed plan additional actions

869 # Shutdown on low battery
870 WHEN UPS-1 REPORTS None->LB : NOTIFY Msg-shutdown NUTLOG Msg-shutdown
871                               WALL MSG-shutdown STARTTIMER final-delay 5
872 WHEN UPS-1 TIMEOUT final-delay : SHUTDOWNCMD "/sbin/shutdown -h 0"
```

Figure 127: Timed shutdown plan, part 2 of 4, the shutdown.

Notes on figure 127

1. Line 852 introduces the notion of "GROUP". In general a group is a set of UPS units which are attached to the same upsd server. In NUT's upsmon.conf the MONITOR *system* declaration identifies the upsd host system and the port. See man upsmon.conf. UPSmon.conf transfers the host system and port identification to a named group, and adds the CERTFILE declaration.
2. Line 853 resembles the upsmon.conf declaration, but with the inclusion of additional keywords for clarification. "UPS-1" declares the UPS name, the HOST and PORT have already been declared. The UPS name should correspond to the name specified in ups.conf. See line 32.
3. Since this is the timed plan rather than the standard plan, we need additional messages which are declared on lines 855-856.
4. When event OL->OB arrives, lines 857-858 call for the "on battery" message to be put on-screen and in the NUT log file. The actions also declare the timers two-min and one-min and start them.
5. When timer one-min runs out, lines 859-863 place warnings on screen, in the NUT log file and on all logged in terminals. The actions also send an email to the administrator.
6. When timer two-min runs out, lines 864-865 place warnings on-screen, in terminals and in the NUT log file. A short final-delay timer is declared and started. This timer corresponds to FINALDELAY in upsmon.conf.
7. What happens if power returns before the shutdown? If event OB->OL arrives, lines 866-867 notify the user, place a message in the NUT log file and turn off all the timers.
8. Whether the plan is "standard" or "timed" the local system must be shutdown on event None ->LB. This happens on lines 870-871. Users receive a final on-screen warning, a message goes into the NUT log file and the action declares and starts a short final-delay timer.
9. When the final-delay timer runs out, line 872 calls for a system shutdown.

I.2.3 Timed shutdown plan, part 3 of 4, warnings

Notes on figure 128

1. Some UPS units are capable of reporting that the battery needs replacement. On line 874, when event None->RB arrives, messages are placed on-screen and in the NUT log file. Line 876 sends an email to the sysadmin. The upsmon RBWARNTIME behaviour is reproduced by defining and starting an rbwarntime timer.
2. Line 880 specifies that when the rbwarntime timer runs out, an on-screen message appears[37] and also goes into the NUT log file. The action also restarts the timer. It will continue to loop until the status [RB] disappears with event RB->None on line 881

[37] Do the users have to be told about this?

```
873 # Warning for battery replacement
874 WHEN UPS-1 REPORTS None->RB : STARTTIMER rbwarntime 43200
875                               NUTLOG Msg-RB NOTIFY Msg-RB
876                               EMAIL FROM « <UPSmon@rogerprice.org> »
877                                     TO   « Roger Price <roger@rogerprice.org> »
878                                     SUBJECT «Msg-RB»
879                                     MESSAGE «Msg-RB»
880 WHEN UPS-1 TIMEOUT rbwarntime : STARTTIMER rbwarntime 43200
                                    NUTLOG Msg-RB NOTIFY Msg-RB
881 WHEN UPS-1 REPORTS RB->None : CANCELTIMER rbwarntime

882 # Warning that UPSmon has lost UPS UPS-1.  Shut down on NOCOMM when OB.
883 WHEN UPS-1 REPORTS COMM->NOCOMM : STARTTIMER nocommwarntime 300
884                                   IF OL->OB NOTIFY Msg-shutdown
885                                   IF OL->OB NUTLOG Msg-shutdown
886                                   IF OL->OB WALL Msg-shutdown
887                                   IF OL->OB STARTTIMER final-delay 5
888 WHEN UPS-1 TIMEOUT nocommwarntime : NUTLOG Msg-NOCOMM NOTIFY Msg-NOCOMM
889 WHEN UPS-1 REPORTS NOCOMM->COMM : CANCELTIMER nocommwarntime
                                   NUTLOG Msg-COMM NOTIFY Msg-COMM
```

Figure 128: Timed shutdown plan, part 3 of 4, warnings,

3. The statuses [COMM] and [NOCOMM] are not due to upsd. They are generated internally by UPSmon.py when it has problems talking to upsd. The `standard` and `timed` configurations discussed here have been tested with upsd and UPSmon.py running in the same machine, but in general this is not the case, and network problems become more apparent when upsd and UPSmon.py are separated.

   The event COMM->NOCOMM starts a timer which will later place a warning message in front of users and in the NUT log file. This follows the upsmon logic. Additionally, and again following upsmon logic, a shutdown procedure will begin if the system is currently running on battery. See lines 884-887. Note that the condition must be attached to each of the actions.

   Note the subtle difference between upsmon and UPSmon.py. See figure 15. On line 68 daemon upsmon will trigger a [NOCOMM] NOTIFY event after `NOCOMMWARNTIME` seconds if it can't reach **any** of the UPS entries in configuration file `upsmon.conf`. UPSmon.py does this for each UPS individually. The difference is slight if there is only one UPS :-)

4. On line 889 the timer `nocommwarntime` is cancelled and suitable messages send to the users[38] and the NUT log file.

[38] Is it really necessary to notify the users of this technical matter?

I.2.4 Timed shutdown plan, part 4 of 4, heartbeat

```
890 # heartbeat.conf
891 # 20 minute heartbeat
892 ups.status: TICK
893 TIMER 600
894 ups.status: TOCK
895 TIMER 600
```

Figure 129: Configuration file heartbeat.conf

The NUT software runs in the background for weeks, months without difficulty and with no messages going the system administrator. "All is well!", but is it?

NUT is a collection of pieces and interconnecting protocols. What if one of these pieces has stopped or the protocol blocked? We need something that will check regularly that all is indeed well. The proposed heartbeat does this job.

Heartbeat definitions are not provided by NUT, you have to create them for yourself. Create the new file heartbeat.conf as shown in figure 129 in the same directory as ups.conf. As called for by line 899, the heartbeat will cycle continuously through this script.

For good security, only users upsd/nut and root should have write access to this file.

```
896 [heartbeat]
897   driver = dummy-ups
898   port = heartbeat.conf
899   mode = dummy-loop
900   desc = "Watch over NUT"
```

Figure 130: Addition to the file ups.conf for heartbeat.conf

Lines 892 and 894 flip the ups.status value between [TICK] and [TOCK].

Lines 893 and 895 place a 10 minute time interval between each status change. $2 \times 600sec = 20min$, the heartbeat period.

You must also declare to upsd that it is to generate the heartbeat. Add the declaration shown in figure 130 to file ups.conf. In line 897 we see the driver used to generate the heartbeat. This driver is also used for debugging. You can amuse yourself by adding further status changes and observing their effect.

Notes on figure 131:

1. On line 904 a group "HB" is declared to contain the heartbeat UPS. The HOST, PORT and CERTFILE are the same as for the physical UPS.
2. Lines 905-906 declare messages specific to the heartbeat.
3. Other than the POWERVAL of 0, the MONITOR declaration on line 907 is the same as for the physical UPS.
4. Line 908 says that the heartbeat does not require electrical energy. This zero declaration also circumvents certain sanity checks that real UPS's must pass.
5. Lines 909 and 912 manage the timers which watch over the [TICK] and [TOCK] coming from upsd. The timer is longer than the expected interval between status arrivals. If this timer expires we assume that the heartbeat has failed.
6. If you uncomment the logging of the None->TICK on line 911 then your log will grow rapidly with a message every 20 minutes.

7. Line 914 is a form of "goto" so all the heartbeart error logging is in one place.
8. Lines 915-919 send heartbeat failure messages to the system administrator and to the NUT log file.

```
901 # Heartbeat operation, requires file heartbeat.conf in the upsd server,
902 # and definition of UPS [heartbeat] in ups.conf.  Note that the timer
903 # specified here must be longer than the timer in heartbeat.conf.
904 GROUP HB HOST localhost PORT 401 CERTFILE Certfile
905 LET Msg-HB-start = banner " Event %(e)s Start HB-timer"
906 LET MSG-HB-fails = banner " %(u)s FAILURE."
                      " I have not received expected TICK/TOCK status change."
907 MONITOR heartbeat POWERVAL 0 UPSDUSER nut-admin PASSWORD «sekret» TYPE primary
908 MINSUPPLIES 0
909 WHEN heartbeat REPORTS None->TICK : CANCELTIMER tock-timer
910                                     STARTTIMER tick-timer 660
911 #                                    NUTLOG Msg-HB-start
912 WHEN heartbeat REPORTS None->TOCK : CANCELTIMER tick-timer
                                        STARTTIMER tock-timer 660

913 # What to do if the heartbeat fails
914 WHEN heartbeat TIMEOUT tick-timer : STARTTIMER tock-timer 0.5
915 WHEN heartbeat TIMEOUT tock-timer : NUTLOG MSG-HB-fails  NOTIFY MSG-HB-fails
916                                     EMAIL FROM « <UPSmon@rogerprice.org> »
917                                           TO   « Price <roger@rogerprice.org> »
918                                           SUBJECT «Msg-HB-fails»
919                                           MESSAGE «Msg-HB-fails»
920 # End of file
```

Figure 131: Timed shutdown plan, part 4 of 4, heartbeat.

I.2.5 Standard shutdown plan

The only differences between the standard plan and the timed shutdown plan are that the standard plan removes lines 854-868 and replaces then with lines 922-923. These actions send a warning message to the users and to the NUT log file.

```
921 # Standard plan specific actions
922 LET Msg-OB = banner " Power failure, possible shutdown, save your work ..."
923 WHEN UPS-1 REPORTS OL->OB :   NOTIFY Msg-OB NUTLOG Msg-OB WALL Msg-OB
924 # End of standard plan specific actions
```

Figure 132: Standard shutdown plan differences

I.3 Redundant power supplies

Please see section 3 and sections "Power values" and "Redundant power supplies" in ***man upsmon****. The upsmon logic is built into the code rather than the configuration file and follows the spirit of the standard shutdown plan preferred by upsmon.*

UPSmon.py allows the system administrator to customise the logic using the configuration file.

I.3.1 MINSUPPLIES failure: Timed shutdown plan

The configuration for a timed shutdown plan for redundant power supplies is very similar to a None ->OB timed shutdown : the status [LS] meaning "Low Supplies" replaces the status [OB]. [LS] says that within a given group, the total powervalue of the UPS units with status [OL] is not sufficient to meet the MINSUPPLIES criterion.

```
925 # Timed shutdown on MINSUPPLIES failure
926 LET Msg-LS = banner " Powervalue failure.  MINSUPPLIES not satisfied."
927 WHEN UPS-1 REPORTS None->LS : NOTIFY Msg-LS    NUTLOG Msg-LS    WALL Msg-LS
928                               EMAIL FROM « <UPSmon@rogerprice.org> »
929                                     TO   « Roger Price <roger@rogerprice.org> »
930                                     SUBJECT «Msg-LS»
931                                     MESSAGE «Msg-LS»
932                               NOTIFY Msg-2-min NUTLOG Msg-2-min WALL Msg-2-min
933                               STARTTIMER two-min 120 STARTTIMER one-min 60
934 WHEN UPS-1 REPORTS LS->None : NOTIFY Msg-OL    NUTLOG Msg-OL    WALL Msg-OL
935                               CANCELTIMER two-min CANCELTIMER one-min
936                               CANCELTIMER final-delay
```

Figure 133: Timed shutdown on MINSUPPLIES failure

I.3.2 MINSUPPLIES failure: Standard shutdown plan

Shutting down a redundant system using the upsmon logic of waiting for [LB] is left as as exercise for the reader. *If that's what you really want, why not go on using upsmon?*



J UPSmon.py installation checklist

Here is the editor's checklist of the things to do to install and run UPSmon.py.

1. Check that you have Python 3.6, or more recent, running. No? You will need to install it.
2. Check that you have OpenSSL 1.1.1d or better.
3. Download UPSmon.py, upsdTLS.py, mkNUTcert.py and mkUPSmonconf.py from rogerprice.org/NUT to wherever you put Python3 scripts.
4. Review the shebangs at the top of the Python3 scripts. Modify if needed to meet the local situation. The shebangs that come with the scripts are those used by the editor. Yours may well be different.
5. Create symlink from `/usr/sbin/UPSmon.py` to wherever you put the Python3 scripts. Create similar links for upsdTLS.py, mkNUTcert.py and mkUPSmonconf.py.
6. Install the systemd service unit `/etc/systemd/system/nut-py-server-shim.service` and the `/etc/systemd/system/nut-py-monitor.service` service unit. See section 12.4.
7. Add programs `shutdown`, `wall` and `notify-send` to `/etc/sudoers` for users `nut`/`upsd`. See section D.2.
8. Run mkNUTcert.py to make TLS certificates. See chapter 10.
9. Run mkUPSmonconf.py to create the UPSmon.py configuration file. See section I.1.
10. Install `/etc/logrotate.d/NUT` . See appendix E,
11. Check that `heartbeat.conf` is installed in the upsd machine and that `ups.conf` contains a `[heartbeat]` declaration.
12. Stop and disable the `nut-monitor` service unit.
13. Run `systemctl daemon-reload` and enable the `nut-py-server-shim` service unit and the `nut-py-monitor` service unit. Start the `nut-py-server-shim` and then the `nut-py-monitor` service units.
14. Check output of command `ps -elf | grep -E "nut|upsd"` which on an openSUSE machine gives the output shown in figure 134.

```
937 1 S upsd 2873 1  9447 -       /usr/lib/ups/driver/usbhid-ups -a Eaton
938 1 S upsd 2878 1  5019 -       /usr/lib/ups/driver/dummy-ups -a heartbeat
939 1 S upsd 2882 1  5017 -       /usr/sbin/upsd
940 5 S upsd 2887 1 17189 core_s  /usr/local/bin/python3.8 -u /usr/sbin/upsdTLS.py
941 5 S upsd 2892 1 58813 -       /usr/local/bin/python3.8 -u /usr/sbin/UPSmon.py
```

Figure 134: upsd and UPSmon.py runtime processes

Questions? Try the `nut-upsuser` mailing list.

Part 5

The End

K Acknowledgments

Editor: As one of the many who have used the work of the NUT project as part of their system setup, I would like to express my gratitude and my appreciation for the software that the NUT project has made available to system administrators through contributions by Charles Lepple, Arjen de Korte, Arnaud Quette, Jim Klimov, Russell Kroll, and many others in the nut-upsuser mailing list.

I would also like to thank those who commented on earlier versions of this text: M.B.M.



L Errors, omissions, obscurities, confusions, typpos...

Please signal errors, omissions, typso and all the other problems you find in this document in the `nut-upsuser` mailing list. Thank you.

Joe's server will still be allright
if power drops off in the night.
That 8 year old pack
of battery back-
up will easily handle th `connection lost`

