NUT
Introduction to Network UPS Tools

Configuration Examples

based on

Network UPS Tools Project 2.7.4

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

with additional text and editing

Roger Price

Version 2018-01-10, with corrections up to 2018-02-04

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.

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.

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 C, "Using notify-send".
- 2018-01-10 Rewrote appendix C, "Using notify-send". Rewrote appendix A "Starting NUT". Added chapter 6.6 "For paranoïd sysadmins".
- 2018-02-04

Contents

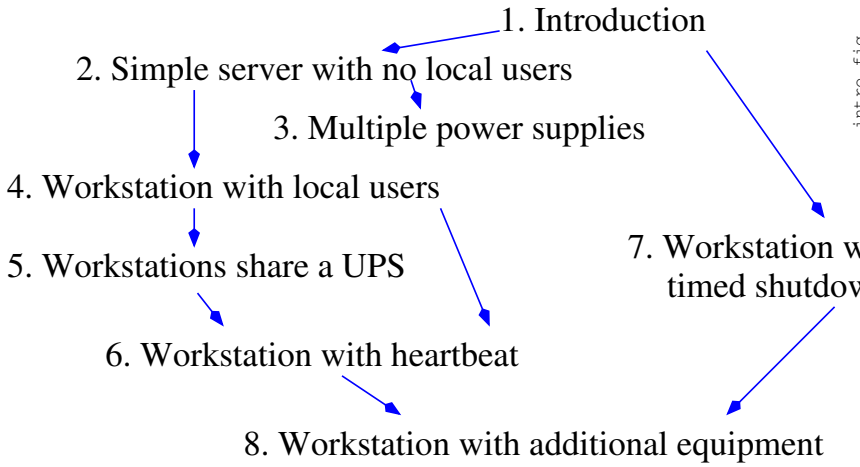
List of Figures

1 Introduction, and Welcome to NUT

This document has been marked up in LaTeX 2ε and rendered as PDF file ConfigExamples.A5.pdf in a portrait A5 format, 74 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. External links may be outlined in cyan, for example `man ups.conf`*. If your mouse hovers over a clickable surface, your browser/PDF reader may tell you where the link leads.*

You are of course free to read as much or as little as you wish of this document, but the suggested reading order is:



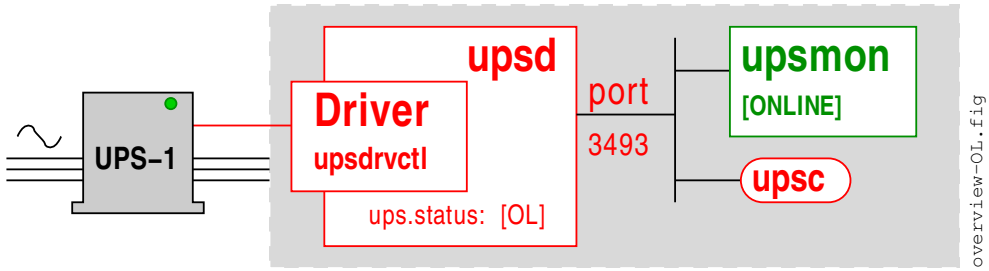


Figure 1: Overview of NUT.

1.1 What is 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 large 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 "ups-user" mailing list[1] in which users may ask questions.

1.2 Why this introduction?

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 Basic components of NUT

Figure 1 shows the basic components of the NUT software.

1.3.1 Driver daemon

The driver is a daemon which 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] See mailing list administration at https://lists.alioth.debian.org/mailman/listinfo/nut-upsuser

1.3.2 Daemon upsd

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[2] and maintains an abstracted image of the UPS in memory[3].

[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 value `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 slave upsmon instances that final shutdown is underway.

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

The various parts of the abstracted image have standardized names, and a key part is `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 typical 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.

Daemon upsd listens on port 3493 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"
```

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

Chapter 1.3.4 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`.

1.3.3 Daemon upsmon

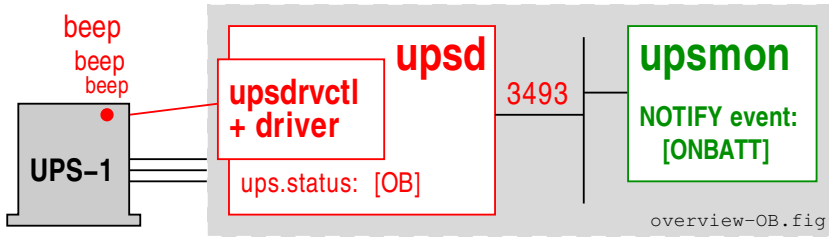


Figure 3: Wall power has failed.

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.

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 master 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.

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]. 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`.

1.3.4 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?"

```
rprice@maria:~> upsc -L
UPS-1: Eaton Ellipse ASR 1500 USBS
heartbeat: Heart beat validation of NUT
```

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

```
rprice@maria:~> upsc UPS-1
battery.charge: 100
battery.charge.low: 50
 ...
driver.name: usbhid-ups
driver.parameter.offdelay: 30
driver.parameter.ondelay: 40
 ...
ups.status: OL CHRG
```

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

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

Man page `man upsc` provides further examples.

1.4 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 managed by upsd.
- `heartbeat.dev` 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".

```
27 password = 123=123          Incorrect
28 password = 123\=123         Good
29 password = "123=123"        Good
```

1.4.1 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.

1.5 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. See "How To Ask Questions The Smart Way" at http://www.catb.org/esr/faqs/smart-questions.html.

If you want to quote configuration files, please remove comments and blank lines. A command such as `grep ^[^#] upsmon.conf` will do the job.

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`.

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.

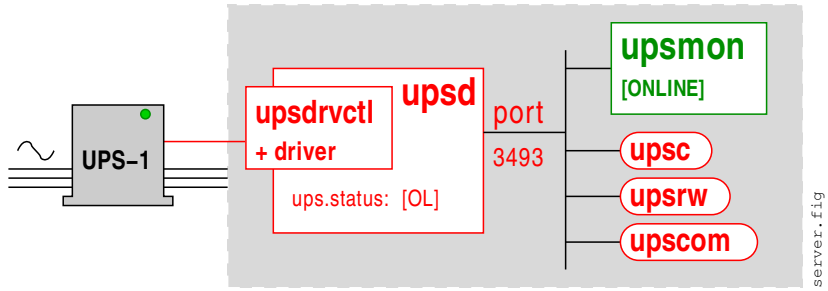


Figure 5: Server with no local users.

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

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

Figure 6: 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 31 begins a UPS-specific section, and names the UPS unit that upsd will manage. The following lines provide details for this 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 in later chapters.

Line 32 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 33 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 34 provides a descriptive text for the UPS.

2.2 Configuration file upsd.conf

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

Figure 7: Configuration file upsd.conf.

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

Line 36 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, his file should be accessible to the upsd process only.

If you do not have IPv6, remove or comment out line 37.

2.3 Configuration file upsd.users

```
38 # upsd.users
39 [upsmaster]
40    password = sekret
41    upsmon master
```

Figure 8: 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 upsd/nut and root can read and write this file.

Line 39 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 40 provides the password. You may prefer something better than "sekret".

Line 41 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 `master` and has authority to shutdown the server. The alternative, "`upsmon slave`", 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`.

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 upsd/nut and root can read and write this file.

```
42 # upsmon.conf
43 MONITOR UPS-1@localhost 1 upsmaster sekret master
```

Figure 9: Configuration file upsmon.conf for a simple server, part 1 of 5.

On line 43

- The UPS name UPS-1 must correspond to that declared in ups.conf line 31.
- The "power value" 1 is the number of power supplies that this UPS feeds on this system.
- upsmaster is the "user" declared in upsd.users line 39.
- sekret is the password declared in upsd.users line 40.
- master means this system will shutdown last, allowing any slaves time to shutdown first. Slave systems will be discussed in chapter 5. There are no slaves in this simple configuration.

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

Figure 10: Configuration file upsmon.conf for a simple server, part 2 of 5.

Line 44 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 45 declares a file created by upsmon when running in master mode when the UPS needs to be powered off. It will be used in more complex configurations. See `man upsmon.conf` for details.

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

Figure 11: Configuration file upsmon.conf for a simple server, part 3 of 5.

Lines 46-55 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

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

Figure 12: Configuration file upsmon.conf for a simple server, part 4 of 5.

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 56-65 declare what is to be done at each NOTIFY event. The declarations, known as "flags" are shown in table 13. You may specify one, two or three flags for each event, in the form `FLAG[+FLAG]*`, however `IGNORE` must always be alone.

`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 13: Flags declaring what upsmon is to do for NOTIFY events.

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.*

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

Figure 14: 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 66 say that this happens every `RBWARNTIME` = 43200 seconds (12 hours).

Line 67: 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 68: When running in master 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 44. 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 UPSs 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

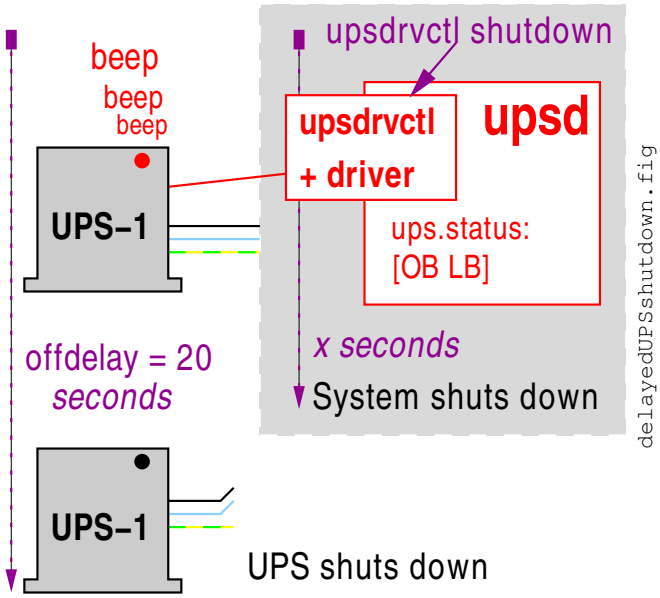


Figure 15: Delayed UPS shutdown.

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 76 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 16 shows the openSUSE adaption of a 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. 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.

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

Figure 16: NUT provided script for delayed UPS shutdown.

The openSUSE distribution places the delayed shutdown script provided by NUT and shown in figure 16 in file `/usr/lib/systemd/system-shutdown/nutshutdown` . The Debian distribution places the script in file `/lib/systemd/system-shutdown/nutshutdown` . In both 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.

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. 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 and issues NOTIFY event [ONBATT]. As instructed by line 57, 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 and issues new NOTIFY event [LOWBATT]. As instructed by line 58 upsmon sends a [LOWBATT] message to syslog and to program `wall`.

6. upsmon decides to command a system shutdown and generates NOTIFY event [SHUTDOWN].

7. upsmon waits `FINALDELAY` seconds as specified on line 68.

8. upsmon creates `POWERDOWN` flag specified on line 45.

9. upsmon calls the `SHUTDOWNCMD` specified on line 44.

10. We now enter the scenario described in figure 15. The operating system's shutdown process takes over. During the system shutdown, the Bash script shown in figure 16 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.

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

12. **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 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 77 in figure 17.

Minutes, hours, days go by...

13. **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.
14. 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.
15. 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].
16. 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 15, 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`.

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

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

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

Figure 17: Configuration file `ups.conf`, improved.

New line 76 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 77 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 78 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[4] 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].
7. upsmon waits `FINALDELAY` seconds as specified on line 68.
8. upsmon creates `POWERDOWN` flag specified on line 45.
9. upsmon calls the `SHUTDOWNCMD` specified on line 44.
10. We now enter the scenario described in figure 15. The operating system's shutdown process takes over. During the system shutdown, the Bash script shown in figure 16 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 .
11. The system powers down before `offdelay` seconds have passed.
12. **Wall power returns before the UPS shuts down** Less than `offdelay` seconds have passed. The UPS continues it's shutdown process.
13. 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*
14. After `ondelay` seconds the UPS turns itself on, and repowers it's outlets
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.

The story continues at state 15 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 31.
For example, to turn on the beeper, use command

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

[4] List needed

where `upsmaster` is the user declared on line 39 and sekret is the l33t password declared on line 40 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 31.
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 upsmaster -p sekret UPS-1@localhost
```

where `upsmaster` is the user declared on line 39 and `sekret` is the password declared on line 40 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 78.

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.

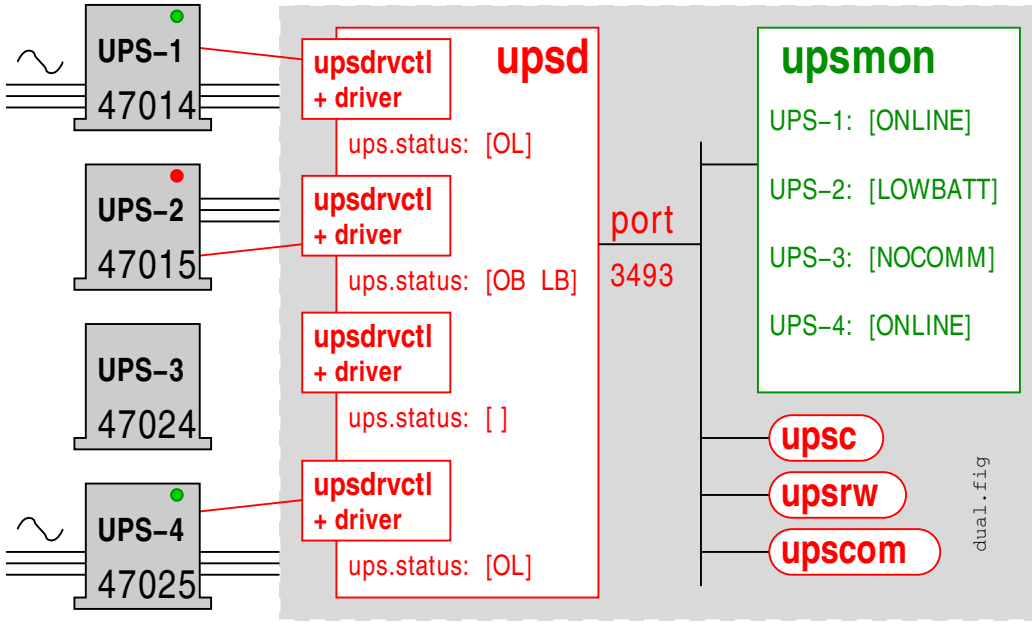


Figure 18: Server with multiple power supplies.

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.

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

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

Figure 19: 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.

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 85, 91, 97 and 103 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 upsd/nut and root can read and write this file.

```
104 # upsmon.conf, multiple power supplies
105 MONITOR UPS-1@localhost 1 upsmaster sekret master
106 MONITOR UPS-2@localhost 1 upsmaster sekret master
107 MONITOR UPS-3@localhost 1 upsmaster sekret master
108 MONITOR UPS-4@localhost 1 upsmaster sekret master
109 MINSUPPLIES 2
```

Figure 20: Configuration file upsmon.conf for multiple power supplies, part 1 of 5.

On lines 105-108

- The UPS names `UPS-1`, `UPS-2`, etc. must correspond to those declared in ups.conf lines 80, 86. 92 and 98.
- The "power value" `1` is the number of power supplies that each UPS feeds on this system.

- `upsmaster` is the "user" declared in `upsd.users` line 39.
- `sekret` is the password declared in `upsd.users` line 40.
- `master` means this system will shutdown last, allowing any slaves time to shutdown first. Slave systems will be discussed in chapter 5. There are no slaves in this configuration.

Line 109, `MINSUPPLIES`, declares that at least two power supplies must be operational, and that if fewer are available, NUT must shut down the server. Figure 18 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, maybe even 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 10-14.

3.3 Shutdown conditions for multiple power supplies

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

Figure 21: 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 53. 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 110-113 in figure 21. Disconnecting the USB cable on a healthy UPS does not cause a system shutdown.

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

Figure 22: Experiment to show effect of lost UPS. Part 2,

However, as shown in figure 22, 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[5] *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.*

[5] "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.

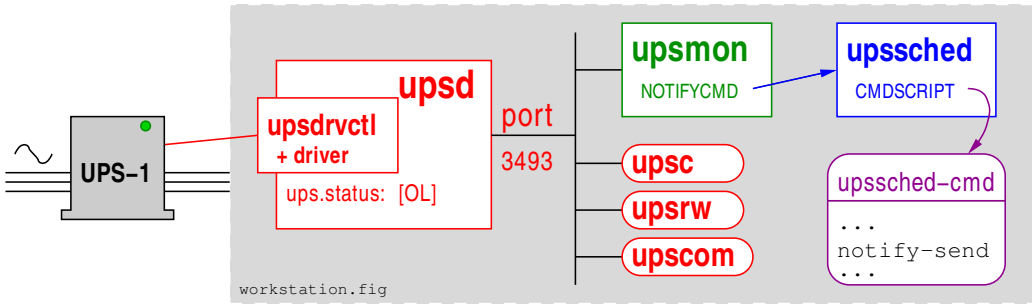


Figure 23: Workstation with local users.

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 seperately 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 seperately in appendix B.

4.1 Configuration file upsmon.conf for a workstation

```
136 # upsmon.conf
137 MONITOR UPS-1@localhost 1 upsmaster sekret master
138 MINSUPPLIES 1
```

Figure 24: 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 upsd/nut and root can read and write this file.

Line 137 is the same as line 43 in the previous chapter.

On line 138, 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.

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

Figure 25: Configuration file upsmon.conf for a workstation, part 2 of 5.

Line 139, identical to line 44 declares the command to be used to shut down the server.

Line 140 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC. Ubuntu sysadmins might see /sbin/upssched.

Line 141, POLLFREQ, declares that the upsmon daemon will poll upsd every 5 seconds.

Line 142, POLLFREQALERT, declares that the upsmon daemon will poll upsd every 5 seconds while the UPS in on battery.

Line 143, HOSTSYNC will be used in master-slave[6] cooperation, to be discussed in chapter 5.4. The default value is 15 seconds.

Line 144 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

[6] A slave 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.

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

Figure 26: Configuration file upsmon.conf for a workstation, part 3 of 5.

The message texts on lines 146-155 in figure 26 do not change.

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

Figure 27: Configuration file upsmon.conf for a workstation, part 4 of 5.

Lines 156-158 now carry the `EXEC` flag: this flag means that when the NOTIFY event occurs, upsmon calls the program identified by the `NOTIFYCMD` on line 140.

Lines 159-165 do not change.

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

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

Lines 166-168 are the same as lines 66-68.

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.

```
169 # upssched.conf
170 CMDSCRIPT /usr/sbin/upssched-cmd
171 PIPEFN /var/lib/ups/upssched.pipe
172 LOCKFN /var/lib/ups/upssched.lock
173
174 AT ONLINE  UPS-1@localhost EXECUTE online
175 AT ONBATT  UPS-1@localhost EXECUTE onbatt
176 AT LOWBATT UPS-1@localhost EXECUTE lowbatt
```

Figure 29: Configuration file upssched.conf for a workstation.

On line 170 CMDSCRIPT points to a user script to be called for designated NOTIFY events. This script will receive as argument a user chosen value. Ubuntu sysadmins might see /usr/local/bin/upssched-script.

Line 171 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 171 the Debian distribution uses /var/run/nut/upssched.pipe.

Here is an example of directory /var/lib/ups taken from distribution openSUSE:

```
177 maria:/ # ls -alF /var/lib/ups
178 drwx------  2 upsd daemon 4096  2 avril 22:53 ./
179 drwxr-xr-x 53 root root   4096 16 mai   01:15 ../
180 -rw-r--r--  1 upsd daemon    6  2 avril 22:48 upsd.pid
181 srw-rw----  1 upsd daemon    0  2 avril 22:53 upssched.pipe=
182 srw-rw----  1 upsd daemon    0  2 avril 22:48 usbhid-ups-UPS-1=
183 -rw-r--r--  1 upsd daemon    6  2 avril 22:48 usbhid-ups-UPS-1.pid
```

Daemon upsmon requires the LOCKFN declaration on line 172 to avoid race conditions. The directory should be the same as PIPEFN.

Line 174 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 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 174 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 175 and 176 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.

```
184 #!/bin/bash -u
185 # upssched-cmd
186 logger -i -t upssched-cmd Calling upssched-cmd $1

187 UPS="UPS-1"
188 STATUS=$( upsc $UPS ups.status )
189 CHARGE=$( upsc $UPS battery.charge )
190 CHMSG="[$STATUS]:$CHARGE%"

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

Figure 30: Configuration script upssched-cmd for a workstation.

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 184 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 186 logs all calls to this script.

Lines 188-190 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 191 the value of the Bash variable `$1` is one of the `EXECUTE` tags defined on lines 174-176.

Lines 192-194 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 198 logs the upssched action, and line 199 calls program notify-send-all to put the message in front of the users. For details of notify-send-all, see appendix C, "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:

```
200 maria:/ # ls -alF /usr/sbin/upssched-cmd
201 -rwxr--r-- 1 upsd daemon 7324  2 avril 16:46 /usr/sbin/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 and issues NOTIFY event [ONBATT]. As instructed by line 157 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 175 to call the user script upssched-cmd with parameter `onbatt`.
5. User script upssched-cmd sees that `$1 = onbatt` and on line 193 sets Bash variable `$MSG` to `UPS-1, [OB DISCHRG]:99% - power failure - save your work!`
6. On line 198, the message is logged, and on line 199 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 and issues new NOTIFY event [LOWBATT]. As instructed by line 158 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 176 to call the user script upssched-cmd with parameter `lowbatt`.
10. User script upssched-cmd sees that `$1 = lowbatt` and on line 194 sets Bash variable `$MSG` to `UPS-1, [OB DISCHRG LB]:12% - shutdown now!`
11. On line 198, the message is logged, and on line 199 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.

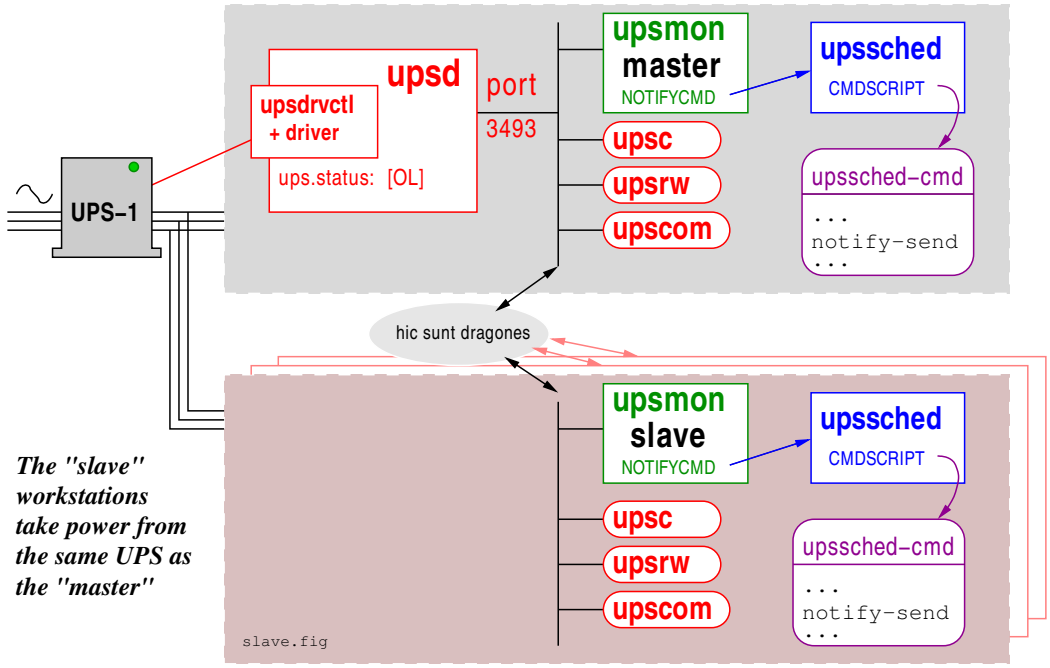


Figure 31: "Slave" workstations take power from same UPS as "master".

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

Figure 31 shows the arrangement. The NUT configuration for the master workstation is identical to that of chapter 4.

Five configuration files specify the operation of NUT in the slave 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 slave. 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 slave

```
202 # upsmon.conf  -- slave --
203 MONITOR UPS-1@master 1 upsmaster sekret slave
204 MINSUPPLIES 1
```

Figure 32: Configuration file upsmon.conf for a slave, part 1 of 5.

This configuration file declares how upsmon in the slave is to handle NOTIFY events coming from the master. For good security, ensure that only users upsd/nut and root can read and write this file.

On line 203

- The UPS name UPS-1 must correspond to that declared in the master ups.conf, line 31. The fully qualified name *UPS@host* includes the network name of the master workstation, in this case master.
- The "power value" 1 is the number of power supplies that this UPS feeds on this system.
- upsmaster is the "user" declared in master upsd.users line 39.
- sekret is the password declared in master upsd.users line 40.
- slave means this system will shutdown first, before the master.

On line 204, 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, man upsmon.conf and file big-servers.txt in the NUT documentation for more details.

```
205 SHUTDOWNCMD "/sbin/shutdown -h +0"
206 NOTIFYCMD /usr/sbin/upssched
207 POLLFREQ 5
208 POLLFREQALERT 5
209 HOSTSYNC 15
210 DEADTIME 15
211 POWERDOWNFLAG /etc/killpower
```

Figure 33: Configuration file upsmon.conf for a slave, part 2 of 5.

Line 205, identical to line 44, declares the command to be used to shut down the slave.

Line 206 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC. Debian administrators would probably specify /sbin/upssched .

Line 207, POLLFREQ, declares that the upsmon daemon will poll upsd in the master every 5 seconds.

Line 208, `POLLFREQALERT`, declares that the upsmon daemon will poll upsd in the master every 5 seconds while the UPS in on battery.

Line 209, `HOSTSYNC` will be used for managing the master-slave shutdown sequence, to be discussed in chapter 5.4. The default value is 15 seconds.

Line 210 specifies how long the slave 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.

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

Figure 34: Configuration file upsmon.conf for a slave, part 3 of 5.

The message texts on lines 212-221 in figure 34 do not change from those in the master.

```
222 NOTIFYFLAG ONLINE   SYSLOG+WALL+EXEC
223 NOTIFYFLAG ONBATT   SYSLOG+WALL+EXEC
224 NOTIFYFLAG LOWBATT  SYSLOG+WALL+EXEC
225 NOTIFYFLAG REPLBATT SYSLOG+WALL
226 NOTIFYFLAG FSD      SYSLOG+WALL
227 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
228 NOTIFYFLAG COMMOK   SYSLOG+WALL
229 NOTIFYFLAG COMMBAD  SYSLOG+WALL
230 NOTIFYFLAG NOCOMM   SYSLOG+WALL
231 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 35: Configuration file upsmon.conf for a slave, part 4 of 5.

Lines 222-224, which do not change from those in the master, carry the `EXEC` flag: when the NOTIFY event occurs, slave upsmon calls the program identified by the `NOTIFYCMD` on line 206.

```
232 RBWARNTIME 43200
233 NOCOMMWARNTIME 300
234 FINALDELAY 5
```

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

Lines 225-231 do not change from those in the master.
Lines 232-234 are the same as lines 66-68 in the master.

5.2 Configuration file upssched.conf for a slave

The NOTIFY events detected by slave 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 master 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.

```
235 # upssched.conf  -- slave  --
236 CMDSCRIPT /usr/sbin/upssched-cmd
237 PIPEFN /var/lib/ups/upssched.pipe
238 LOCKFN /var/lib/ups/upssched.lock
239
240 AT ONLINE  UPS-1@master EXECUTE online
241 AT ONBATT  UPS-1@master EXECUTE onbatt
242 AT LOWBATT UPS-1@master EXECUTE lowbatt
```

Figure 37: Configuration file upssched.conf for a slave.

On line 236, CMDSCRIPT points to a user script to be called for designated NOTIFY events. This script will receive as argument a user chosen value.

Line 237 defines PIPEFN which is the file name of a socket used for communication between upsmon and upssched. As in the master, it is important that the directory be accessible to NUT software and nothing else. The value shown in figure 37 is for the openSUSE distribution. Debian uses /var/run/nut/upssched.pipe.

Daemon upsmon requires the LOCKFN declaration on line 238 to avoid race conditions. The directory should be the same as PIPEFN.

Line 240 says what is to be done by upssched for NOTIFY event [ONLINE]. The "UPS-1@master" says that it applies to the UPS controlled by the master, and the EXECUTE says that the user script specified by CMDSCRIPT is to be called with argument "online".

Lines 241 and 242 make similar declarations for NOTIFY events [ONBATT] and [LOWBATT].

5.3 Configuration script upssched-cmd for a slave

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.

```
243 #!/bin/bash -u
244 # upssched-cmd  --slave --
245 logger -i -t upssched-cmd Calling upssched-cmd $1
```

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

Figure 38: Configuration script upssched-cmd for a slave.

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 243 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 245 logs all calls to this script.

On line 246 the value of the Bash variable `$1` is one of the `EXECUTE` tags defined on lines 240-242.

Lines 247-249 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 slave. Accented letters and non latin characters are allowed.

Line 253 logs the upssched action, and line 254 calls program notify-send-all to put the message in front of the slave users. For details of notify-send-all, see appendix C, "Using notify-send". See also `notify-send --help`. There is no man page.

5.4 Magic: How does the master shut down the slaves?

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

When the master makes the shutdown decision, it places a status symbol [FSD] in the abstract image of the UPS maintained by it's upsd. The slave upsmon daemons poll the master upsd every `POLLFREQ` seconds as delared on line 141, and when they see the [FSD] symbol, knowing that they are a slave, they shut down immediately. The master waits for the slaves to react and shutdown. The waiting period is specified by `HOSTSYNC` on line 143. After this time has elapsed, the master will shut down, even if there is a slave 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 slave systems from getting stuck if the master fails to respond in time. After a UPS becomes critical, the slave will wait up to `HOSTSYNC` seconds for the master to set the [FSD] flag. If that timer expires, the slave will assume that the master is broken and will shut down anyway. See also `man upsmon.conf`.

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.

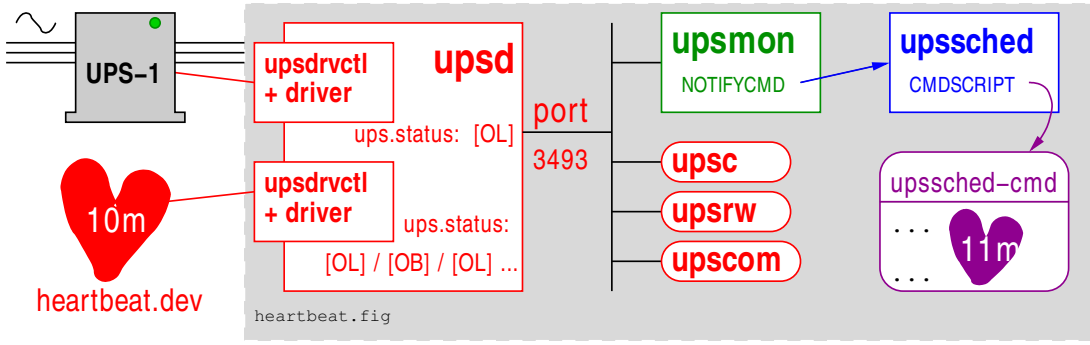


Figure 39: Workstation with heartbeat.

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.dev` 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.

```
255 # ups.conf, heartbeat
256 [UPS-1]
257    driver = usbhid-ups
258    port = auto
259    desc = "Eaton ECO 1600"
260    offdelay = 60
261    ondelay = 70
262    lowbatt = 33

263 [heartbeat]
264    driver = dummy-ups
265    port = heartbeat.dev
266    desc = "Watch over NUT"
```

Figure 40: Configuration file ups.conf for workstation with heartbeat.

Lines 256-262 are unchanged.

New line 263 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 264 says that this UPS is of type dummy-ups, i.e. a software UPS, for which the behaviour will be in a file specified by the port declaration.

Line 265 says that the behaviour is in file heartbeat.dev in the same directory as ups.conf. It is traditional in NUT that such files have file type .dev .

See man dummy-ups for lots of details.

6.2 Configuration file heartbeat.dev for workstation

```
267 # heartbeat.dev -- 10 minute heartbeat
268 ups.status: OL
269 TIMER 300
270 ups.status: OB
271 TIMER 300
```

Figure 41: Configuration file heartbeat.dev for workstation.

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

The dummy UPS will cycle continuously through this script.

Lines 268 and 270 flip the ups.status value between [OL] and [OB].

Lines 269 and 271 place a 5 minute time interval between each status change. $2 \times 300sec = 10min$, 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.

```
272 # upsmon.conf
273 MONITOR UPS-1@localhost     1 upsmaster sekret master
274 MONITOR heartbeat@localhost 0 upsmaster sekret master
275 MINSUPPLIES 1
```

Figure 42: Configuration file upsmon.conf for a workstation with heartbeat.

The change is the addition of line 274 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:

```
276 NOTIFYFLAG ONLINE   EXEC
277 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.

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`. 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.

```
278 # Restart timer which completes only if the dummy-ups heart beat
279 # has stopped.  See timer values in heartbeat.dev
280 AT ONBATT heartbeat@localhost CANCEL-TIMER heartbeat-failure-timer
281 AT ONBATT heartbeat@localhost START-TIMER  heartbeat-failure-timer 660
```

Figure 43: 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 280, 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 281, 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

```
282 AT ONLINE * ...
283 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 301.

On lines 289 and 290, change the e-mail addresses to something that works for you.

Lines 301-308 introduce the `heartbeat-failure-timer` case into the case statement. Line 302 specifies a message to be logged with the current UPS status as defined on lines 292-295.

Lines 304-306 compose a message to the sysadmin which is sent on line 307. The message includes the current state of those NUT kernel processes which are operational.

```
#!/bin/bash -u
# upssched-cmd for workstation with heartbeat
logger -i -t upssched-cmd Calling upssched-cmd $1

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

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

case $1 in
(online)  MSG="$UPS, $CHMSG - power supply had been restored." ;;
(onbatt)  MSG="$UPS, $CHMSG - power failure - save your work!" ;;
(lowbatt) MSG="$UPS, $CHMSG - shutdown now!" ;;
(heartbeat-failure-timer)
   MSG="NUT heart beat fails. $CHMSG" ;;
   # Email to sysadmin
   MSG1="Hello, upssched-cmd reports NUT heartbeat has failed."
   MSG2="Current status: $CHMSG \n\n$0 $1"
   MSG3="\n\n$( ps -elf | grep -E 'ups[dms]|nut' )"
   echo -e "$MSG1 $MSG2 $MSG3" | /bin/mail -r "$EMAIL_FROM" \
        -s "NUT heart beat fails. Currently $CHMSG" "$EMAIL_TO"
(*) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
      exit 1 ;;
esac
logger -i -t upssched-cmd $MSG
notify-send-all "$MSG"
```

Figure 44: Configuration script upssched-cmd including heartbeat.

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:

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

Figure 45: Heartbeat watcher.

Line 315 specifies who is the owner of the upssched process.

Line 319 will succeed if there is a process managing the heartbeat.

Lines 321, 322 and 325 show three different ways of telling the sysadmin that all is well with the heartbeat process. Pick which one(s) suit you. See appendix C for a discussion of notify-send-all.

The Bash script requires something like line 333 in `/etc/crontab`:

```
333 1 8 * * *  upsd  /usr/local/bin/heartbeat-watcher.sh > /dev/null 2>&1
```

In this example, line 333 declares that the Bash script is to be run at 08:01 hrs every day as user “upsd”. Debian would use “nut”. See 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".

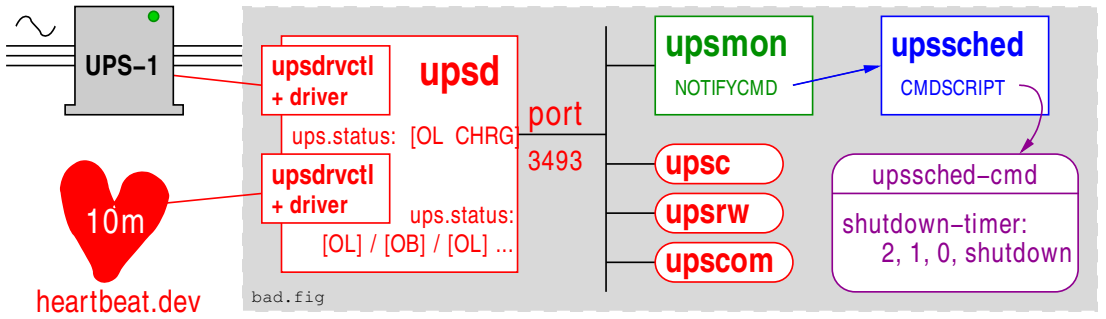


Figure 46: Workstation with timed shutdown.

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.dev` 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

```
334 # ups.conf, timed shutdown
335 [UPS-1]
336    driver = usbhid-ups
337    port = auto
338    desc = "Eaton ECO 1600"
339    offdelay = 60
340    ondelay = 70
341    lowbatt = 33
342
343 [heartbeat]
344    driver = dummy-ups
345    port = heartbeat.dev
346    desc = "Watch over NUT"
```

Figure 47: 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 335 and 343 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 336 and 344 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 337 and 345 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.

Lines 338 and 346 provide descriptive texts for the UPS.

For a detailed discussion of `offdelay` and `ondelay` on lines 339-340, see chapter 2.7.

Additional line 341 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[7] will restore the default, so if you want a permanent change you must change the default. See also chapter 2.10.

[7] List needed

7.2 Configuration file heartbeat.dev for workstation with timed shutdown

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

```
347 # heartbeat.dev -- 10 minute heartbeat
348 ups.status: OL
349 TIMER 300
350 ups.status: OB
351 TIMER 300
```

Figure 48: Configuration file heartbeat.dev for workstation with timed shutdown.

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

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

The dummy UPS will cycle continuously through this script.

Lines 348 and 350 flip the ups.status value between [OL] and [OB].

Lines 349 and 351 place a 5 minute time interval between each status change. $2 \times 300sec = 10min$, the heartbeat period.

7.3 Configuration file upsd.conf with timed shutdown

```
352 # upsd.conf
353 LISTEN 127.0.0.1 3493
354 LISTEN ::1 3493
```

Figure 49: Configuration file upsd.conf or 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 36-37.

Line 353 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 354.

7.4 Configuration file upsd.users with timed shutdown

```
355 # upsd.users
356 [upsmaster]
357    password = sekret
358    upsmon master
```

Figure 50: 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 39-41. For good security, ensure that only users upsd/nut and root can read and write this file.

Line 356 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 357 provides the password. You may prefer something better than "sekret".

Line 358 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 `master`.

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.

```
359 # upsmon.conf
360 MONITOR UPS-1@localhost     1 upsmaster sekret master
361 MONITOR heartbeat@localhost 0 upsmaster sekret master
362 MINSUPPLIES 1
```

Figure 51: 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 upsd/nut and root can read and write this file.

On line 360

- The UPS name `UPS-1` must correspond to that declared in ups.conf line 335.
- The "power value" `1` is the number of power supplies that this UPS feeds on this system.
- `upsmaster` is the "user" declared in upsd.users line 39.
- `sekret` is the password declared in upsd.users line 40.
- `master` means this system will shutdown last, allowing any slaves time to shutdown first. There are no slaves in this simple configuration.

Line 361 declares that upsmon is also to monitor the heartbeat.

On line 362, `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.

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

Figure 52: Configuration file `upsmon.conf` with timed shutdown, part 2 of 5.

Line 363 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.

Line 364 says which program is to be invoked when upsmon detects a NOTIFY event flagged as `EXEC`. Debian and Ubuntu sysadmins might see `/sbin/upssched`.

Line 365, `POLLFREQ`, declares that the upsmon daemon will poll upsd every 5 seconds.

Line 366, `POLLFREQALERT`, declares that the upsmon daemon will poll upsd every 5 seconds while the UPS in on battery.

Line 367, `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 368, `POWERDOWNFLAG` declares a file created by upsmon when running in master mode when the UPS needs to be powered off. It will be used in more complex configurations. See `man upsmon.conf` for details.

Lines 369-378 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

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

Figure 53: Configuration file upsmon.conf with timed shutdown, part 3 of 5.

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 364.

```
379 NOTIFYFLAG ONLINE   EXEC
380 NOTIFYFLAG ONBATT   EXEC
381 NOTIFYFLAG LOWBATT  SYSLOG+WALL
382 NOTIFYFLAG REPLBATT SYSLOG+WALL
383 NOTIFYFLAG FSD      SYSLOG+WALL
384 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
385 NOTIFYFLAG COMMOK   SYSLOG+WALL
386 NOTIFYFLAG COMMBAD  SYSLOG+WALL
387 NOTIFYFLAG NOCOMM   SYSLOG+WALL
388 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 54: Configuration file upsmon.conf with timed shutdown, part 4 of 5.

Lines 379-388 declare what is to be done at each NOTIFY event. The declarations, known as "flags" are shown in table 13. You may specify one, two or three flags for each event, in the form FLAG[+FLAG]*, however IGNORE must always be alone.

Lines 379-380 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 364.

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.*

When a UPS says that it needs to have its battery replaced, upsmon will generate a [REPLBATT] NOTIFY event. Line 389 say that this happens every RBWARNTIME = 43200 seconds (12 hours).

Line 390: 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.

```
389 RBWARNTIME 43200
390 NOCOMMWARNTIME 300
391 FINALDELAY 5
```

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

Line 391: When running in master 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 363. 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 UPSs 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.

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.

```
392 # upssched.conf
393 CMDSCRIPT /usr/sbin/upssched-cmd
394 PIPEFN /var/lib/ups/upssched.pipe
395 LOCKFN /var/lib/ups/upssched.lock
396
397 AT ONBATT UPS-1@localhost START-TIMER two-minute-warning-timer 5
398 AT ONBATT UPS-1@localhost START-TIMER one-minute-warning-timer 65
399 AT ONBATT UPS-1@localhost START-TIMER shutdown-timer 125
400
401 AT ONLINE UPS-1@localhost CANCEL-TIMER two-minute-warning-timer
402 AT ONLINE UPS-1@localhost CANCEL-TIMER one-minute-warning-timer
403 AT ONLINE UPS-1@localhost CANCEL-TIMER shutdown-timer
404 AT ONLINE UPS-1@localhost EXECUTE ups-back-on-line
405
406 AT ONBATT heartbeat@localhost CANCEL-TIMER heartbeat-failure-timer
407 AT ONBATT heartbeat@localhost START-TIMER  heartbeat-failure-timer 660
```

Figure 56: Configuration file upssched.conf with timed shutdown.

On line 393 CMDSCRIPT points to a user script to be called for designated NOTIFY events. This script will receive as argument a user chosen timer name. Ubuntu sysadmins might see /usr/local/bin/upssched-script.

Line 394 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 394 the Debian distribution uses /var/run/nut/upssched.pipe.

Here is an example of directory /var/lib/ups taken from distribution openSUSE:

```
408 drwx------  2 upsd daemon 4096 24 mai   11:04 ./
409 drwxr-xr-x 53 root root   4096 24 mai   01:15 ../
410 srw-rw----  1 upsd daemon    0 20 mai   23:13 dummy-ups-heartbeat=
411 -rw-r--r--  1 upsd daemon    5 20 mai   23:13 dummy-ups-heartbeat.pid
412 -rw-r--r--  1 upsd daemon    5 20 mai   23:13 upsd.pid
413 srw-rw----  1 upsd daemon    0 24 mai   11:04 upssched.pipe=
414 srw-rw----  1 upsd daemon    0 20 mai   23:13 usbhid-ups-UPS-1=
415 -rw-r--r--  1 upsd daemon    5 20 mai   23:13 usbhid-ups-UPS-1.pid
```

Daemon upsmon requires the LOCKFN declaration on line 395 to avoid race conditions. The directory should be the same as PIPEFN.

Line 397 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 397 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 398 and 399 do the same thing for the 65 second timer one-minute-warning-timer and the 125 second timer shutdown-timer.

Line 401 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 402 and 403 do the same thing for the 65 second timer "one-minute-warning-timer" and the 125 second timer "shutdown-timer".

Line 404 command `EXECUTE` says that upssched is to call the user script immediately with the argument "`ups-back-on-line`".

On line 406, 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 407, 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

```
416 #!/bin/bash -u
417 # upssched-cmd  Workstation with heartbeat and timed shutdown
418 logger -i -t upssched-cmd Calling upssched-cmd $1

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

422 UPS="UPS-1"
423 STATUS=$( upsc $UPS ups.status )
424 CHARGE=$( upsc $UPS battery.charge )
425 CHMSG="[$STATUS]:$CHARGE%"
```

Figure 57: 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 420 and 421, change the e-mail addresses to something that works for you.

Lines 422-425 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.

Lines 427-433 introduce the `heartbeat-failure-timer` case into the case statement. Line 428 specifies a message to be logged with the current UPS status as defined on lines 422-425.

Lines 429-431 compose a message to the sysadmin which is sent on line 432. The message includes the current state of those NUT kernel processes which are operational.

```
426 case $1 in
427 (heartbeat-failure-timer)
428    MSG="NUT heart beat fails. $CHMSG" ;;
429    MSG1="Hello, upssched-cmd reports NUT heartbeat has failed."
430    MSG2="Current status: $CHMSG \n\n$0 $1"
431    MSG3="\n\n$( ps -elf | grep -E 'ups[dms]|nut' )"
432    echo -e "$MSG1 $MSG2 $MSG3" | /bin/mail -r "$EMAIL_FROM" \
433         -s "NUT heart beat fails. Currently $CHMSG" "$EMAIL_TO" ;;
```

```
434 (two-minute-warning-timer)
435    MSG="Possible shutdown in 2 minutes. Save your work! $CHMSG" ;;
436 (one-minute-warning-timer)
437    MSG="Probable shutdown in 1 minute. Save your work! $CHMSG" ;;
438 (shutdown-timer)
439    MSG="Power  failure shutdown: Calling upsmon -c fsd, $CHMSG" ;;
440    /usr/sbin/upsmon -c fsd ;;
441 (ups-back-on-line)
442    MSG="Power back, shutdown cancelled. $CHMSG"  ;;
443 (*) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
444       exit 1 ;;
445 esac
446 logger -i -t upssched-cmd $MSG
447 notify-send-all "$MSG"
```

Figure 58: Configuration script upssched-cmd for timed shutdown, 2 of 2.

7.7.1 The timed shutdown

The cases at lines 434 and 436 specify warnings to be notified to the users when the `two-minute-warning-timer` and `one-minute-warning-timer` complete.

Beginning at line 438 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 440 that we use upsmon to shut down the system. This automatically takes into account any slave systems which need to be shut down as well.

Line 441 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 C 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 and issues NOTIFY event [ONBATT]. As instructed by line 380 upsmon calls upssched, specified by `NOTIFYCMD` on line 364. 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 397-399. 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 393 with the timer name as argument. In the script, this matches the case on line 434 which defines a suitable warning message in Bash variable `MSG`. Line 446 logs this message and line 447 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 436 which defines a stronger warning message in Bash variable `MSG`. Line 446 logs this message and line 447 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 438 which defines an ultimate warning message in Bash variable `MSG`, and then calls upsmon for a system shutdown. Line 446 logs message `MSG` and line 447 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 391.
10. upsmon creates `POWERDOWN` flag specified on line 368.
11. upsmon calls the `SHUTDOWNCMD` specified on line 363.

12. We now enter the scenario described in figure 15. The operating system's shutdown process takes over. During the system shutdown, the Bash script shown in figure 16 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 339.
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 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.

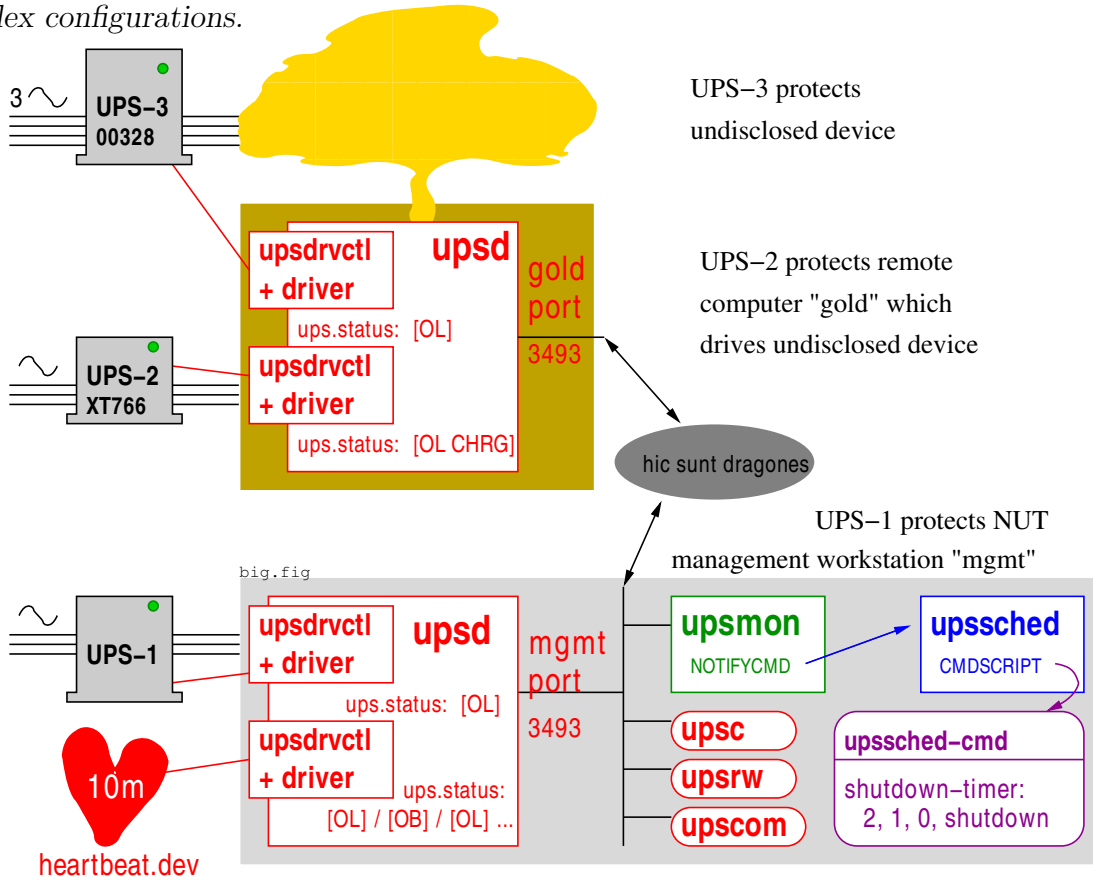


Figure 59: Workstation with additional equipment.

The configuration is for an industrial application in which some unspecified industrial equipment is protected by a UPS, and is also driven by a computer system having it's own UPS. This equipment with the driving computer is at a remote site, code name gold. Overall management is from a computer at a different 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.dev`: 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

```
448 # nut.conf  -- gold --
449 MODE=netserver
```

Figure 60: File `nut.conf` for gold.

mgmt

```
450 # nut.conf  -- mgmt --
451 MODE=standalone
```

Figure 61: 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 449 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.dev

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

gold

```
452 # ups.conf  -- gold --
453 [UPS-3]
454    driver = usbhid-ups
455    port = auto
456    desc = "Huge 3 phase"
457    offdelay = 20
458    ondelay = 30
459    lowbatt = 33
460    serial = 00328
461
462 [UPS-2]
463    driver = usbhid-ups
464    port = auto
465    desc = "Small monophase"
466    offdelay = 20
467    ondelay = 30
468    lowbatt = 33
469    serial = XT766
```

Figure 62: File ups.conf for gold.

mgmt

```
470 # ups.conf  -- mgmt --
471 [UPS-1]
472    driver = usbhid-ups
473    port = auto
474    desc = "Eaton ECO 1600"
475    offdelay = 60
476    ondelay = 70
477    lowbatt = 33
478
479 [heartbeat]
480    driver = dummy-ups
481    port = heartbeat.dev
482    desc = "Watch over NUT"
```

Figure 63: File ups.conf for mgmt.

```
483 # heartbeat.dev -- 10 min
484 ups.status: OL
485 TIMER 300
486 ups.status: OB
487 TIMER 300
```

Figure 64: heartbeat.dev for mgmt.

gold: On lines 453-462 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 463 and 454 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 457-458 and 466-467 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 460 and 469. See `man usbhid-ups`.

mgmt: On lines 471-476 we offer a specimen definition for `UPS-1` and on lines 484-487 we propose the dummy UPS "heartbeat" discussed in chapter 6. The heartbeat requires the definition file heartbeat.dev, lines 484-487, to be placed in the same directory as ups.conf.

8.3 Configuration files upsd.conf

gold

```
488 # upsd.conf   -- gold --
489 LISTEN 10.8.0.5 3493
490 LISTEN X::Y::Z 3493
```

Figure 65: File upsd.conf for gold.

mgmt

```
491 # upsd.conf   -- mgmt --
492 LISTEN 127.0.0.1 3493
493 LISTEN ::1 3493
```

Figure 66: 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 489 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 489 and 490 for your own needs.

mgmt: Line 492 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 490 and 493.

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

8.4 Configuration files upsd.users

gold

```
494 # upsd.users   -- gold --
495 [upsmaster]
496    password = sekret
497    upsmon master
```

Figure 67: File upsd.users for gold.

mgmt

```
498 # upsd.users   -- mgmt --
499 [upsmaster]
500    password = sekret
501    upsmon master
```

Figure 68: 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 upsd/nut 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 495 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 496 provides the password. You may prefer something better than "sekret".

Line 497 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 slave" 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 499 declares the "user name" of the system administrator who has write access to UPS-1 and to the heartbeat managed by upsd.

Line 500 provides another uberl33t password.

Line 501 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.

```
502 # upsmon.conf  -- mgmt --
503 MONITOR UPS-3@gold          0 upsmaster sekret master
504 MONITOR UPS-2@gold          0 upsmaster sekret master
505 MONITOR UPS-1@localhost     1 upsmaster sekret master
506 MONITOR heartbeat@localhost 0 upsmaster sekret master
507 MINSUPPLIES 1
```

Figure 69: 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 upsd/nut and root can read and write this file.

Line 503 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 467.
- 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.
- upsmaster is the "user" declared in upsd.users line 495.
- sekret is the l33t password declared in upsd.users line 496.
- master means this system will shutdown last, allowing any slaves time to shutdown first. There are no slaves on gold.

Line 504 specifies that upsmon on mgmt will also monitor UPS-2 which supplies the gold computer.

Line 505 specifies that upsmon on mgmt will monitor UPS-1 which supplies power to mgmt itself. Note the "power value" of 1.

Line 506 declares that upsmon is also to monitor the heartbeat.

On line 507, `MINSUPPLIES` sets the number of power supplies that must be receiving power to keep the mgmt 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.

```
508 SHUTDOWNCMD "/sbin/shutdown -h +0"
509 NOTIFYCMD /usr/sbin/upssched
510 POLLFREQ 5
511 POLLFREQALERT 5
512 DEADTIME 15
513 POWERDOWNFLAG /etc/killpower
```

Figure 70: Configuration file `upsmon.conf` for mgmt, part 2 of 5.

Line 508 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.

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

Line 509 says which program is to be invoked when upsmon detects a NOTIFY event flagged as `EXEC`.

Line 510, `POLLFREQ`, declares that the upsmon daemon will poll upsd in gold and in mgmt every 5 seconds.

Line 511, `POLLFREQALERT`, declares that the upsmon daemon will poll the upsd daemons every 5 seconds while any UPS in on battery.

Line 512, `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 513, `POWERDOWNFLAG` declares a file created by upsmon when running in master mode when `UPS-1` needs to be powered off. See `man upsmon.conf` for details.

Lines 514-523 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

```
514 NOTIFYMSG ONLINE   "UPS %s: On line power."
515 NOTIFYMSG ONBATT   "UPS %s: On battery."
516 NOTIFYMSG LOWBATT  "UPS %s: Battery is low."
517 NOTIFYMSG REPLBATT "UPS %s: Battery needs to be replaced."
518 NOTIFYMSG FSD      "UPS %s: Forced shutdown in progress."
519 NOTIFYMSG SHUTDOWN "Auto logout and shutdown proceeding."
520 NOTIFYMSG COMMOK   "UPS %s: Communications (re-)established."
521 NOTIFYMSG COMMBAD  "UPS %s: Communications lost."
522 NOTIFYMSG NOCOMM   "UPS %s: Not available."
523 NOTIFYMSG NOPARENT "upsmon parent dead, shutdown impossible."
```

Figure 71: Configuration file upsmon.conf for mgmt, part 3 of 5.

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 509.

```
524 NOTIFYFLAG ONLINE   EXEC
525 NOTIFYFLAG ONBATT   EXEC
526 NOTIFYFLAG LOWBATT  SYSLOG+WALL
527 NOTIFYFLAG REPLBATT SYSLOG+WALL
528 NOTIFYFLAG FSD      SYSLOG+WALL
529 NOTIFYFLAG SHUTDOWN SYSLOG+WALL
530 NOTIFYFLAG COMMOK   SYSLOG+WALL
531 NOTIFYFLAG COMMBAD  SYSLOG+WALL
532 NOTIFYFLAG NOCOMM   SYSLOG+WALL
533 NOTIFYFLAG NOPARENT SYSLOG+WALL
```

Figure 72: Configuration file upsmon.conf for mgmt, part 4 of 5.

Lines 524-533 declare what is to be done at each NOTIFY event. The declarations, known as "flags" are shown in table 13. You may specify one, two or three flags for each event, in the form `FLAG[+FLAG]*`, however `IGNORE` must always be alone.

Lines 524-525 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 509.

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.*

When a UPS says that it needs to have its battery replaced, upsmon will generate a [REPLBATT] NOTIFY event. Line 534 say that this happens every `RBWARNTIME` = 43200 seconds (12 hours).

```
534 RBWARNTIME 43200
535 NOCOMMWARNTIME 300
536 FINALDELAY 5
```

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

Line 535: 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 536: When running in master 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 363. 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 UPSs 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.

8.6 Configuration file upssched.conf for mgmt

Daemon upsmon in mgmt detects the NOTIFY events due to status changes in gold and 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 538 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 539 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 539 the Debian distribution uses /var/run/nut/upssched.pipe.

Daemon upsmon requires the LOCKFN declaration on line 540 to avoid race conditions. The directory should be the same as PIPEFN.

8.6.1 UPS-3 on gold

Lines 542 and 543 say what is to be done by upssched for a NOTIFY event [ONBATT] due to UPS-3 on gold. On line 542 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 543 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

```
537 # upssched.conf  -- mgmt --
538 CMDSCRIPT /usr/sbin/upssched-cmd
539 PIPEFN /var/lib/ups/upssched.pipe
540 LOCKFN /var/lib/ups/upssched.lock
541
542 AT ONBATT UPS-3@gold      START-TIMER UPS-3-two-minute-warning-timer 5
543 AT ONBATT UPS-3@gold      START-TIMER UPS-3-shutdown-timer 125
544 AT ONLINE UPS-3@gold      CANCEL-TIMER UPS-3-two-minute-warning-timer
545 AT ONLINE UPS-3@gold      CANCEL-TIMER UPS-3-shutdown-timer
546 AT ONLINE UPS-3@gold      EXECUTE UPS-3-back-on-line
547
548 AT ONBATT UPS-2@gold      START-TIMER UPS-2-two-minute-warning-timer 5
549 AT ONBATT UPS-2@gold      START-TIMER UPS-2-shutdown-timer 125
550 AT ONLINE UPS-2@gold      CANCEL-TIMER UPS-2-two-minute-warning-timer
551 AT ONLINE UPS-2@gold      CANCEL-TIMER UPS-2-shutdown-timer
552 AT ONLINE UPS-2@gold      EXECUTE UPS-2-back-on-line
553
554 AT ONBATT UPS-1@localhost START-TIMER UPS-1-two-minute-warning-timer 5
555 AT ONBATT UPS-1@localhost START-TIMER UPS-1-shutdown-timer 125
556 AT ONLINE UPS-1@localhost CANCEL-TIMER UPS-1-two-minute-warning-timer
557 AT ONLINE UPS-1@localhost CANCEL-TIMER UPS-1-shutdown-timer
558 AT ONLINE UPS-1@localhost EXECUTE UPS-1-back-on-line
559
560 AT ONBATT heartbeat@localhost CANCEL-TIMER heartbeat-failure-timer
561 AT ONBATT heartbeat@localhost START-TIMER  heartbeat-failure-timer 660
```

Figure 74: Configuration file `upssched.conf` for mgmt.

544-546 say what is to be done by upssched for NOTIFY event [ONLINE]. The `CANCEL-TIMER` 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 546 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 548 and 549 define the timers which start when upssched receives a NOTIFY event [ONBATT], and lines 550 and 551 cancel those timers when hopefully upssched receives NOTIFY event [ONLINE].

Line 552 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 554 and 555 define the timers which start when upssched receives a NOTIFY event [ONBATT], and lines 556 and 557 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 558 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 560, 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 561, 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

```
562 #!/bin/bash -u
563 # upssched-cmd  -- mgmt --
564 logger -i -t upssched-cmd Calling upssched-cmd $1
565
566 # Send emails to/from these addresses
567 EMAIL_TO="sysadmin@example.com"
568 EMAIL_FROM="upssched-cmd@${HOSTNAME:-nut}.example.com"
569
570 function make-STCH {
571    STCH="[$( upsc $1 ups.status )]:$( upsc $1 battery.charge )%"}
572 case $1 in
```

Figure 75: 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 75, on lines 567 and 568, change the e-mail addresses to something that works for you.

Lines 570-571 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 572 covering all the possible parameter values (timer names) that the user script may expect.

```
573 (UPS-3-two-minute-warning-timer)  make-STCH UPS-3@gold
574    MSG="UPS-3: gold power failure. $STCH" ;;
575 (UPS-3-shutdown-timer)            make-STCH UPS-3@gold
576    MSG="UPS-3: gold shutdown. $STCH" ;;
577                 Commands for undisclosed device shutdown
578 (UPS-3-back-on-line)              make-STCH UPS-3@gold
579    MSG="UPS-3: power returns. $STCH"  ;;

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

Figure 76: User script upssched-cmd on mgmt, 2 of 5.

In figure 76, lines 573-579 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 577 will shut down the undisclosed device attached to gold.

```
581 (UPS-1-two-minute-warning-timer)  make-STCH UPS-1
582    MSG="UPS-1: gold power failure. $STCH" ;;
583 (UPS-1-shutdown-timer)            make-STCH UPS-1
584    MSG="UPS-1: gold shutdown. $STCH" ;;
585    /usr/sbin/upsmon -c fsd ;;
586 (UPS-1-back-on-line)              make-STCH UPS-1
587    MSG="UPS-1: power returns. $STCH"  ;;
```

Figure 77: User script upssched-cmd on mgmt, 3 of 5.

In figure 77, lines 581-587 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 code on line 585 will shut down the workstation.

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

Figure 78: User script upssched-cmd on mgmt, 4 of 5.

In figure 78, lines 588-594 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 590, 591 and 592 prepare a message for the sysadmin in Bash variables `MSG1`, `MSG2` and `MSG3`. Lines 593-594 e-mail the message to the sysadmin. The message includes the current state of those NUT kernel processes which are operational.

```
595 (*) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
596       exit 1 ;;
597 esac
598 logger -i -t upssched-cmd $MSG
599 notify-send-all "$MSG"
```

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

In figure 79, lines 595-596 cover any unexpected parameter values, and lines 598-599 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.

9 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, Russell Kroll, and many others in the nut-upsuser mailing list.

I would also like to thank those who commented on early versions of this text.

10 Errors, omissions, obscurities, confusions, typpos...

Please signal errors, omissions, typso and all the other problems you find in this document in the "ups-user" mailing list[8]. 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`

[8]See mailing list administration at https://lists.alioth.debian.org/mailman/listinfo/nut-upsuser

Appendix

A Starting NUT

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

Figure 80: 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

Daemon	systemd service unit	Notes
driver	`nut-driver.service`	One or more driver daemons as specified in file `ups.conf`. This service unit is started by systemd whenever `nut-server.service` starts.
upsd	`nut-server.service`	The central daemon which maintains the abstracted view of the UPS units.
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 specified by the `NOTIFYCMD` command in `upsmon.conf`.

Figure 81: 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 [9] 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[10].

- `MODE=standalone` This is the most common situation in which line 600 in figure 80 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`.

[9] See script `/sbin/upsd`.
[10] See script `/sbin/upsmon`.

B Stopping NUT

B.1 Delayed UPS shutdown with NUT script

We saw in chapter 2, line 44, 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 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 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 76 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 openSUSE distribution places the delayed shutdown script provided by NUT and shown in figure 82 in file `/usr/lib/systemd/system-shutdown/nutshutdown` . The Debian distribution places the script in file `/lib/systemd/system-shutdown/nutshutdown` .

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

Figure 82: UPS shutdown script `nutshutdown`.

On line 604 the call to upsmon with option `-K` checks the `POWERDOWNFLAG` defined by line 45. The upsmon daemon creates this file when running in 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 606.

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

Figure 83: UPS shutdown script `nutshutdown` for 2 of 3 UPS's.

See also `man upsdrvctl`

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 84.

```
607 # nut-delayed-ups-shutdown.service
608 [Unit]
609   Description=Initiate delayed UPS shutdown
610   Before=umount.target
611   DefaultDependencies=no
612 [Service]
613   Type=oneshot
614   ExecStart=/usr/bin/logger -t nut-delayed-ups-shutdown\
                                "upsdrvctl shutting down UPS"
615   ExecStart=/sbin/upsdrvctl shutdown   # Debian
616 [Install]
617   WantedBy=final.target
```

Figure 84: UPS shutdown service unit `nut-delayed-ups-shutdown.service`.

The `ExecStart` directive on line 615 will shutdown [11] 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 615 you should specify the required UPS's as shown in lines 618-619.

```
618   ExecStart=/sbin/upsdrvctl shutdown UPS-2   # Debian
619   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`. 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`.

[11] 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 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.

C.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`. None of these programs has a man page and the editor has not been able to find a mailing list specific to desktop notifications.

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 a 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.

C.2 Give user "upsd" ("nut") 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 "upsd" or "nut". We will use the name "upsd". "upsd" 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`

```
620 # Host alias specification
621 Host_Alias LAN = 10.218.0/255.255.255.0,127.0.0.1,localhost,maria
622
623 uspd  LAN = (ALL)  NOPASSWD:SETENV: /usr/bin/notify-send
```

Figure 85: Modifications to file `/etc/sudoers`

Line 621 corresponds to the editor's system and should be adapted to your setup.
On line 623 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.

C.3 Search for and notify logged in users

Figure 86 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`.

```
624 #! /bin/bash -u
625 # notify-send-all sends notifications to all X displays
626 # Assumes /etc/sudoers allows caller to sudo as any user.
627 # E.g. nut LAN = (ALL) NOPASSWD:SETENV: /usr/bin/notify-send
628 # Call with text to be displayed as argument.
629 XUSERS=( $( who | grep -E "\(:[0-9](\.[0-9])*\)" \
630          | awk '{print $1$NF}' | sort -u ) )
631 for XUSER in $XUSERS      # E.g. jschmo(:0)
632 do NAME=(${XUSER/\(/ })   # Insert space, make NAME an array
633    DISPLAY=${NAME[1]/)/}  # E.g. :0
634    sudo -u ${NAME[0]} DISPLAY=${DISPLAY} \
635             /usr/bin/notify-send -t 0 -u critical "$@"; RC=$?
636    if [[ $RC -ne 0 ]]; then exit $RC; fi
637 done
```

Figure 86: Bash script notify-send-all

Line 629 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 632 creates a Bash array containing the user name and the X display number in the form `jschmo :0)`.

Line 633 extracts the X display number `:0` and on line 634 calls notify-send to notify the user as if user "upsd" ("nut" on Debian) 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.

C.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 199.

While wall power is disconnected, use a command such as `ps -elf | grep -E "ups[dms]|nut"` to find the programs running as user "upsd" ("nut" on Debian):

```
638 upsd  2635     1  ... /usr/bin/usbhid-ups -a Eaton
639 upsd  2637     1  ... /usr/bin/dummy-ups -a heartbeat
640 upsd  2641     1  ... /usr/sbin/upsd
641 root  2645     1  ... /usr/sbin/upsmon
642 upsd  2646  2645  ... /usr/sbin/upsmon
643 upsd  3217     1  ... /usr/sbin/upssched UPS Eaton@localhost: On battery
644 upsd  3236     1  ... dbus-launch --autolaunch=d1cd...ca5d2 ...
645 upsd  3237     1  ... /bin/dbus-daemon --fork --print-pid 5 ...
646 upsd  3241     1  ... /usr/lib/xfce4/notifyd/xfce4-notifyd
647 upsd  3243     1  ... /usr/lib/xfce4/xfconf/xfconfd
```

Lines 638-643 are due to NUT activity, and lines 644-647 are due to the use of notify-send. Note on line 646 that the `xfce4-notifyd` daemon is running as user "upsd"!

C.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 https://unix.stackexchange.com/questions/2881/show-a-notification-across-all-running-x-displays
3. The Gnome "Desktop Notifications Specification" is still a very long way from being RFC quality.

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, RedHat or Ubuntu based systems, using other desktop environments such as Cinnamon, KDE or Gnome.