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 2020-11-27, with corrections up to 2020-11-27

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.

The source file for this document has been marked up by the editor in LaTeX 2ε and rendered as PDF file ConfigExamples.A5.pdf in a portrait A5 format, 134 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.*

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 23, "Using notify-send".
- 2018-01-10 Rewrote appendix 23, "Using notify-send". Rewrote appendix 20 "Starting NUT". Added chapter 6.6 "For paranoïd sysadmins".
- 2018-08-22 In chapter 3.1 added reference to issue #597 for multiple UPS units.
- 2019-07-21 Added chapter 9 "Encrypted connections".
- 2020-08-20 File `heartbeat.dev` becomes `heartbeat.conf`
- 2020-09-30 Added Part 2 covering the Python3 scripts. Deprecated 9 "Encrypted connections".
- 2020-11-27

Contents

List of Figures

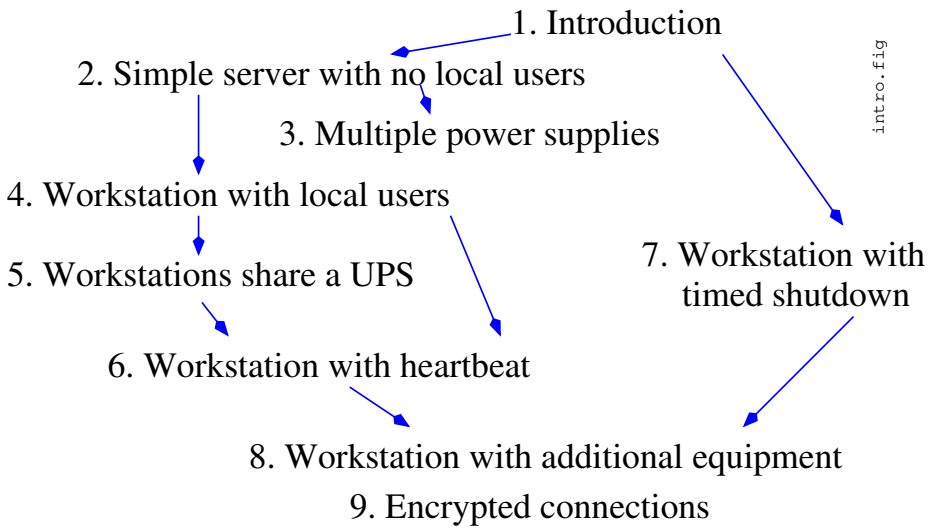
Part 1

UPS monitoring using NUT

The first part of this documentation discusses UPS activity monitoring using the facilities provided by NUT 2.7.4. Part 2 will discuss the use of the UPSmon.py software to manage the UPS activity. Part 3 provides technical appendices.

1 Introduction, and Welcome to NUT

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



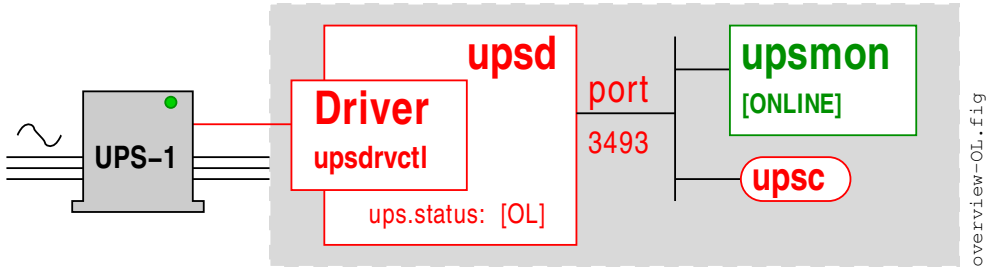


Figure 1: Overview of NUT.

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 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.

1.3.2 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[1] and maintains an abstracted image of the UPS in memory[2].

[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.

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

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"
```

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

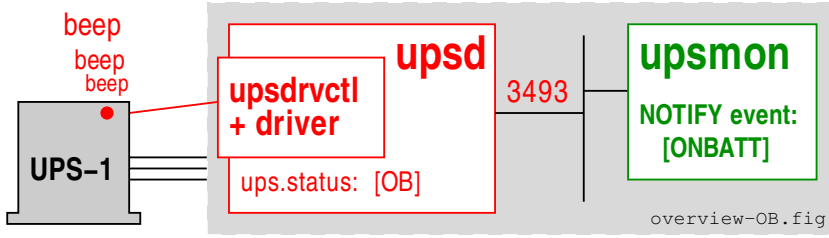


Figure 3: Wall power has failed.

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.

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?"

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.

```
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.conf` Used only for heartbeat configurations.
- `upsd.conf` Access control to the upsd daemon.
- `upsd.users` Who has access to the upsd daemon.
- `upsmon.conf` upsmon daemon configuration.
- `upssched.conf` Only used for customised and timer-based setups.
- **upssched-cmd** A script used only for customised and timer-based setups.
- **delayed UPS shutdown** Choice of scripts for delayed UPS shutdown.

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

First, most of the programs use an uppercase word to declare a configuration directive. This may be something like `MONITOR`, `NOTIFYCMD`, or `ACCESS`. Case matters here. "`monitor`" won't be recognized.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Much better.

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

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

1.4.1 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 which you will find in "How To Ask Questions The Smart Way" at http://www.catb.org/esr/faqs/smart-questions.html.

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

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

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.

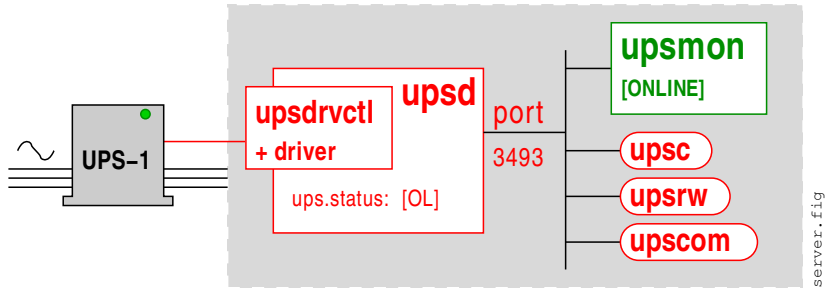


Figure 5: Server with no local users.

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 20.
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 21.

2.1 Configuration file `ups.conf`, first attempt

```
31 # ups.conf, first attempt
32 [UPS-1]
33    driver = usbhid-ups
34    port = auto
35    desc = "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 32 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 33 specifies the driver that upsd will use. For the full list of drivers, see the Hardware Compatibility list and the required drivers at http://www.networkupstools.org/stable-hcl.html.

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

Line 35 provides a descriptive text for the UPS.

2.2 Configuration file upsd.conf

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

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

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

2.3 Configuration file upsd.users

```
39 # upsd.users
40 [upsmaster]
41    password = sekret
42    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 40 declares the "user name" of the system administrator who has write access to the UPS's managed by upsd. It is independent of `/etc/passwd`. The upsmon client daemon will use this name to poll and command the UPS's. There may be several names with different levels of access. For this example we only need one.

Line 41 provides the password. You may prefer something better than "sekret".

Line 42 declares that this user is the upsmon daemon, and the required set of actions will be set automatically. In this simple configuration daemon upsmon is a `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.

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

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

On line 44

- The UPS name UPS-1 must correspond to that declared in ups.conf line 32.
- The "power value" 1 is the number of power supplies that this UPS feeds on this system.
- upsmaster is the "user" declared in upsd.users line 40.
- sekret is the password declared in upsd.users line 41.
- 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.

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

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

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

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

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

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

Lines 47-56 assign a text message to each NOTIFY event. Within each message, the marker %s is replaced by the name of the UPS which has produced this event. upsmon passes this message to program wall to notify the system administrator of the event. You can change the default

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

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

messages to something else if you like. The format is NOTIFYMSG event "*message*" where %s is replaced with the identifier of the UPS in question.

Lines 57-66 declare what is to be done at each NOTIFY event. The declarations, known as "flags" are shown in table 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.*

```
67 RBWARNTIME 43200
68 NOCOMMWARNTIME 300
69 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 67 say that this happens every RBWARNTIME = 43200 seconds (12 hours).

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

Line 69: When running in 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 45. If you need to let your users do something in between those events, increase this number. Remember, at this point your UPS battery is almost depleted, so don't make this too big. Alternatively, you can set this very low so you don't wait around when it's time to shut down. Some UPS's don't give much warning for low battery and will require a value of 0 here for a safe shutdown.

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

2.5 The delayed UPS shutdown

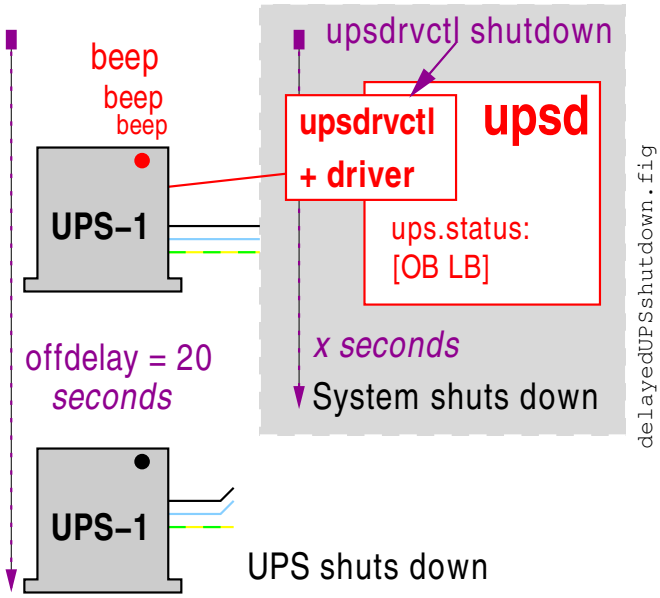


Figure 15: Delayed UPS shutdown.

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 77 if you're curious.)

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

Figure 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.

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

```
70  #!/bin/sh
71  #/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 21 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 58, an [ONBATT] message goes to syslog and to program `wall`. The server is still operational running on the UPS battery.

   Minutes go by...

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

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

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

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

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

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

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 78 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.

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

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

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

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

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

Additional line 79 sets the default value for `battery.charge.low`. Even if you use command upsrw to set a value for `battery.charge.low`, usbhid-ups and some other drivers[3] 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 69.
8. upsmon creates `POWERDOWN` flag specified on line 46.
9. upsmon calls the `SHUTDOWNCMD` specified on line 45.
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 32.
For example, to turn on the beeper, use command

[3] List needed

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

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

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

2.10 Utility program upsrw

Utility program upsrw is a command line program for changing the values of UPS variables. To see which variables may be changed, type `upsrw` *`ups-name`* where *`ups-name`* is the name of the UPS as declared in file `ups.conf`, line 32.

For example, at line 9 we saw that the `battery.charge.low` has been set to 50. We will change this to something less conservative with command

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

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

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

which returns the value 33.

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

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

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

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

Chapter 4 will show how to overcome these difficulties.

3 Server with multiple power supplies

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

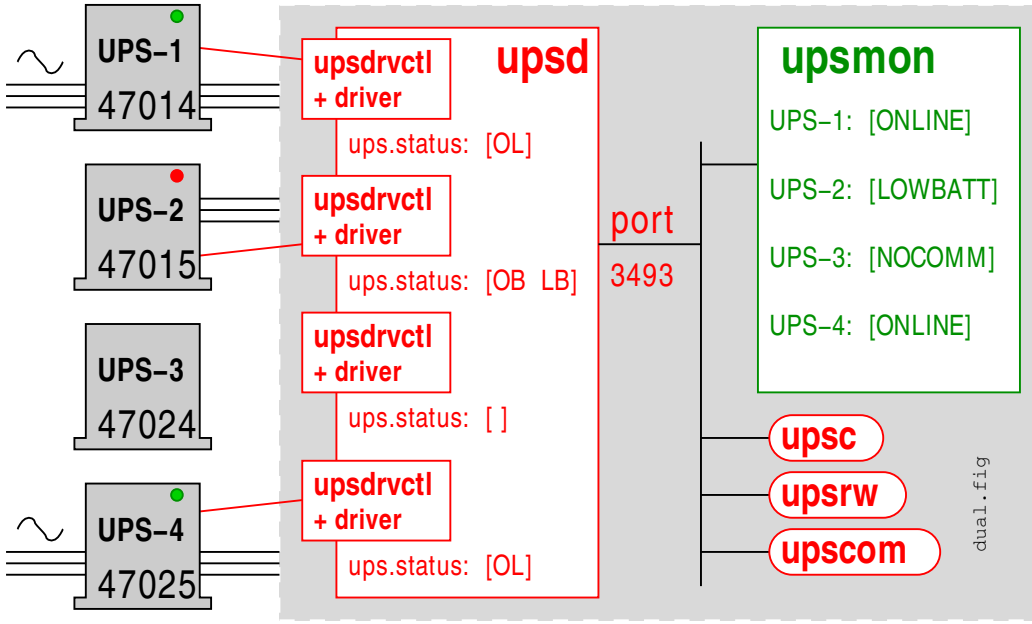


Figure 18: Server with multiple power supplies.

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 20.
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 21.

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

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

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

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

Figure 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. For oher drivers, which do not provide the ability to distinguish UPS units, or for UPS units which have no serial number, see the comment by Charles Lepple in NUT issue #597 at https://github.com/networkupstools/nut/issues/597.

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

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

3.2 Configuration file upsmon.conf for multiple power supplies

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

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

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

On lines 106-109

- The UPS names `UPS-1`, `UPS-2`, etc. must correspond to those declared in ups.conf lines 81, 87. 93 and 99.

- The "power value" `1` is the number of power supplies that each UPS feeds on this system.
- `upsmaster` is the "user" declared in `upsd.users` line 40.
- `sekret` is the password declared in `upsd.users` line 41.
- `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 110, `MINSUPPLIES`, declares that at least two power supplies must be operational, and that if fewer are available, NUT must shut down the server. Figure 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

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

Figure 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 54. However this does not count as an [LB].

You can verify this yourself on a simple working configuration such as that of chapter 2 using the Bash command shown on lines 111-114 in figure 21. Disconnecting the USB cable on a healthy UPS does not cause a system shutdown.

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

Figure 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[4] *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.*

[4] "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.

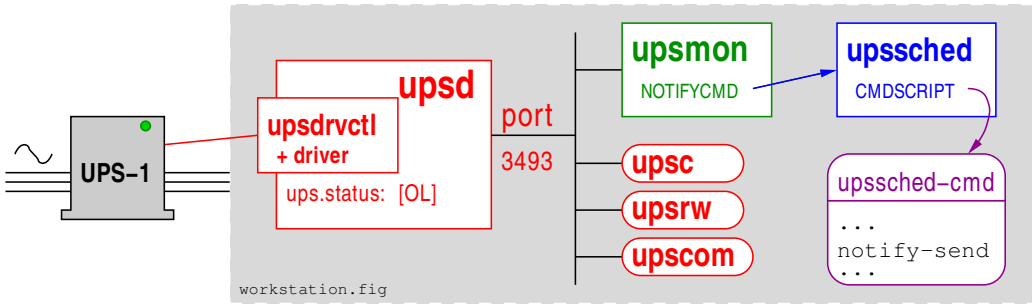


Figure 23: Workstation with local users.

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 20.
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 21.

4.1 Configuration file upsmon.conf for a workstation

```
137 # upsmon.conf
138 MONITOR UPS-1@localhost 1 upsmaster sekret master
139 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 138 is the same as line 44 in the previous chapter.

On line 139, MINSUPPLIES sets the number of power supplies that must be receiving power to keep this system running. 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.

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

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

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

Line 141 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC. Ubuntu sysadmins might see /sbin/upssched.

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

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

Line 144, HOSTSYNC will be used in master-slave[5] cooperation, to be discussed in chapter 5.4. The default value is 15 seconds.

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

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

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

[5] 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.

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

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

The message texts on lines 147-156 in figure 26 do not change.

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

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

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

Lines 160-166 do not change.

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

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

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

4.2 Configuration file upssched.conf for a workstation

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

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

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

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

On line 171 CMDSCRIPT points to a user script to be called for designated NOTIFY events. This script will receive as argument a user chosen value. Ubuntu sysadmins might see /usr/local/bin/upssched-script.

Line 172 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 172 the Debian distribution uses /var/run/nut/upssched.pipe.

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

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

Daemon upsmon requires the LOCKFN declaration on line 173 to avoid race conditions. The directory should be the same as PIPEFN.

Line 175 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 175 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 176 and 177 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.

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

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

```
192 case $1 in
193    online) MSG="$UPS, $CHMSG - power supply has been restored." ;;
194    onbatt) MSG="$UPS, $CHMSG - power failure - save your work!" ;;
195    lowbatt) MSG="$UPS, $CHMSG - shutdown now!" ;;
196    *) logger -i -t upssched-cmd "Bad arg: \"$1\", $CHMSG"
197       exit 1 ;;
198 esac
199 logger -i -t upssched-cmd $MSG
200 notify-send-all "$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 185 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 187 logs all calls to this script.

Lines 189-191 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 192 the value of the Bash variable `$1` is one of the `EXECUTE` tags defined on lines 175-177.

Lines 193-195 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 199 logs the upssched action, and line 200 calls program notify-send-all to put the message in front of the users. For details of notify-send-all, see appendix 23, "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:

```
201 maria:/ # ls -alF /usr/sbin/upssched-cmd
202 -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 158 an [ONBATT] message goes to syslog, to program `wall` and to upssched. The server is still operational, running on the UPS battery.
4. upssched ignores the message it receives and follows the instruction on line 176 to call the user script upssched-cmd with parameter `onbatt`.
5. User script upssched-cmd sees that `$1 = onbatt` and on line 194 sets Bash variable `$MSG` to `UPS-1, [OB DISCHRG]:99% - power failure - save your work!`
6. On line 199, the message is logged, and on line 200 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 159 upsmon sends a [LOWBATT] message to syslog, to program `wall` and to upssched.

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

9. upssched ignores the message it receives and follows the instruction on line 177 to call the user script upssched-cmd with parameter `lowbatt`.
10. User script upssched-cmd sees that `$1 = lowbatt` and on line 195 sets Bash variable `$MSG` to `UPS-1, [OB DISCHRG LB]:12% - shutdown now!`
11. On line 199, the message is logged, and on line 200 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.

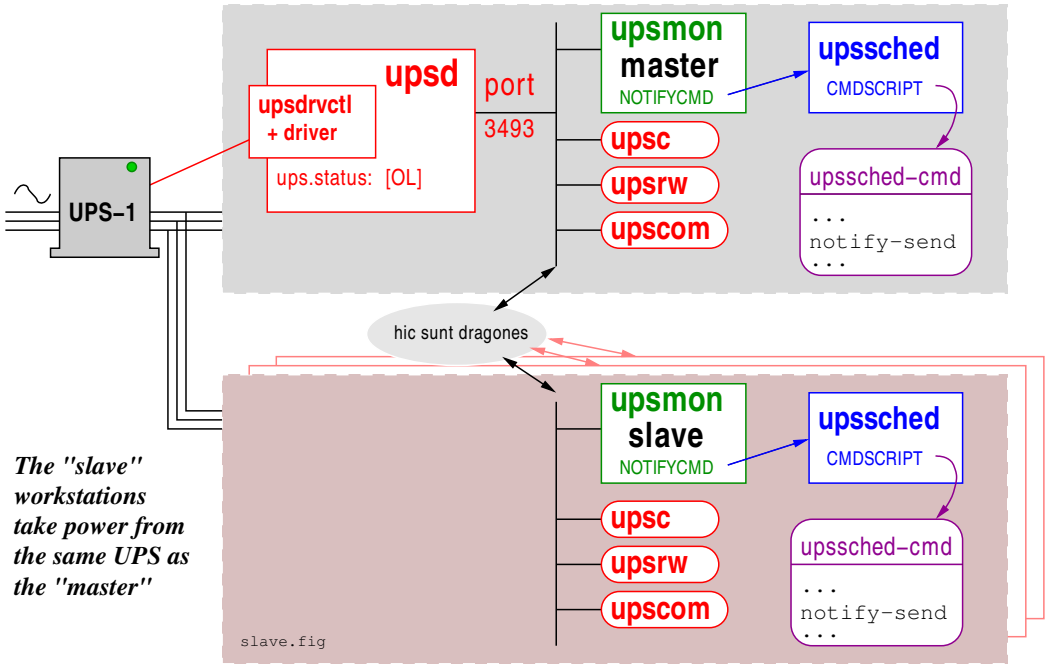


Figure 31: “Slave” workstations take power from same UPS as “master”.

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 20.
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 21.

5.1 Configuration file upsmon.conf for a slave

```
203 # upsmon.conf  -- slave --
204 MONITOR UPS-1@master 1 upsmaster sekret slave
205 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 204

- The UPS name UPS-1 must correspond to that declared in the master ups.conf, line 32. 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 40.
- sekret is the password declared in master upsd.users line 41.
- slave means this system will shutdown first, before the master.

On line 205, 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.

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

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

Line 206, identical to line 45, declares the command to be used to shut down the slave.

Line 207 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC. Debian administrators would probably specify /sbin/upssched .

Line 208, POLLFREQ, declares that the upsmon daemon will poll upsd in the master every 5 seconds.

Line 209, `POLLFREQALERT`, declares that the upsmon daemon will poll upsd in the master every 5 seconds while the UPS in on battery.

Line 210, `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 211 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.

```
213 NOTIFYMSG ONLINE   "UPS %s: On line power."
214 NOTIFYMSG ONBATT   "UPS %s: On battery."
215 NOTIFYMSG LOWBATT  "UPS %s: Battery is low."
216 NOTIFYMSG REPLBATT "UPS %s: Battery needs to be replaced."
217 NOTIFYMSG FSD      "UPS %s: Forced shutdown in progress."
218 NOTIFYMSG SHUTDOWN "Auto logout and shutdown proceeding."
219 NOTIFYMSG COMMOK   "UPS %s: Communications (re-)established."
220 NOTIFYMSG COMMBAD  "UPS %s: Communications lost."
221 NOTIFYMSG NOCOMM   "UPS %s: Not available."
222 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 213-222 in figure 34 do not change from those in the master.

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

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

Lines 223-225, 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 207.

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

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

Lines 226-232 do not change from those in the master.
Lines 233-235 are the same as lines 67-69 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.

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

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

On line 237, CMDSCRIPT points to a user script to be called for designated NOTIFY events. This script will receive as argument a user chosen value.

Line 238 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 239 to avoid race conditions. The directory should be the same as PIPEFN.

Line 241 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 242 and 243 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.

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

```
247 case $1 in
248    online) MSG="UPS-1 - power supply had been restored." ;;
249    onbatt) MSG="UPS-1 - power failure - save your work!" ;;
250    lowbatt) MSG="UPS-1 - shutdown now!" ;;
251    *) logger -i -t upssched-cmd "Bad arg: \"$1\""
252       exit 1 ;;
253 esac
254 logger -i -t upssched-cmd $MSG
255 notify-send-all "$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 244 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 246 logs all calls to this script.

On line 247 the value of the Bash variable `$1` is one of the `EXECUTE` tags defined on lines 241-243.

Lines 248-250 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 254 logs the upssched action, and line 255 calls program notify-send-all to put the message in front of the slave users. For details of notify-send-all, see appendix 23, "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 142, 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 144. 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.

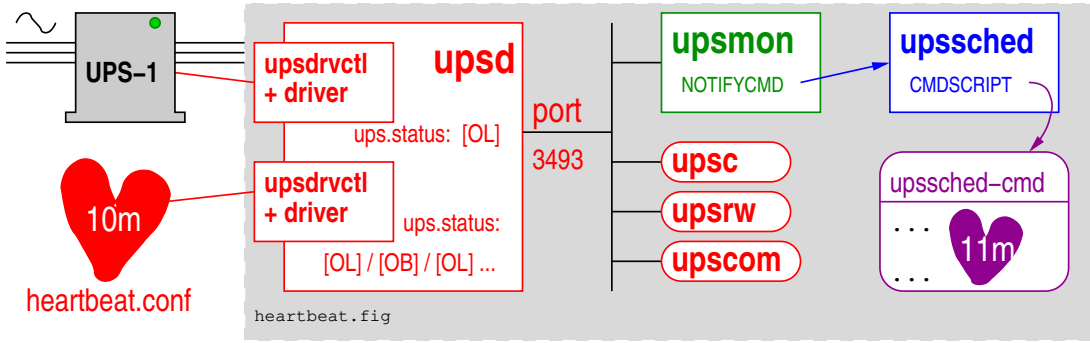


Figure 39: Workstation with heartbeat.

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 20.
2. The upsd UPS declarations: `ups.conf` will be extended to include the heartbeat. See chapter 6.1.
3. New configuration file `heartbeat.conf` defines the dummy UPS which provides the heartbeat. See chapter 6.2.
4. The upsd daemon access control: File `upsd.conf` as given in chapter 2.2 stays the same.
5. The upsd user declarations: File `upsd.users` as given in chapter 2.3 does not change.
6. The upsmon daemon configuration: `upsmon.conf`. See chapter 6.3.
7. The upssched configuration: `upssched.conf`. See chapter 6.4.
8. The upssched-cmd script: see chapter 6.5.

9. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix 21.

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.

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

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

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

Lines 257-263 are unchanged.

New line 264 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 265 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 266 says that the behaviour is in file `heartbeat.conf` 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.conf for workstation

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

Figure 41: Configuration file heartbeat.conf for workstation.

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

The dummy UPS will cycle continuously through this script.

Lines 269 and 271 flip the ups.status value between [OL] and [OB].

Lines 270 and 272 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.

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

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

The change is the addition of line 275 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:

```
277 NOTIFYFLAG ONLINE   EXEC
278 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.

```
279 # Restart timer which completes only if the dummy-ups heart beat
280 # has stopped.  See timer values in heartbeat.conf
281 AT ONBATT heartbeat@localhost CANCEL-TIMER heartbeat-failure-timer
282 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 281, 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 282, 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

```
283 AT ONLINE * ...
284 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 302.

On lines 290 and 291, change the e-mail addresses to something that works for you.

Lines 302-309 introduce the `heartbeat-failure-timer` case into the case statement. Line 303 specifies a message to be logged with the current UPS status as defined on lines 293-296.

Lines 305-307 compose a message to the sysadmin which is sent on line 308. 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:

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

Figure 45: Heartbeat watcher.

Line 316 specifies who is the owner of the upssched process. See table 126 for a list of possible owners.

Line 320 will succeed if there is a process managing the heartbeat.

Lines 322, 323 and 326 show three different ways of telling the sysadmin that all is well with the heartbeat process. Pick which one(s) suit you. See appendix 23 for a discussion of notify-send-all.

The Bash script requires something like line 334 in `/etc/crontab`:

```
334 1 8 * * *  upsd  /usr/local/bin/heartbeat-watcher.sh > /dev/null 2>&1
```

In this example, line 334 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)`. See table 126 for a list of possible users.

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".

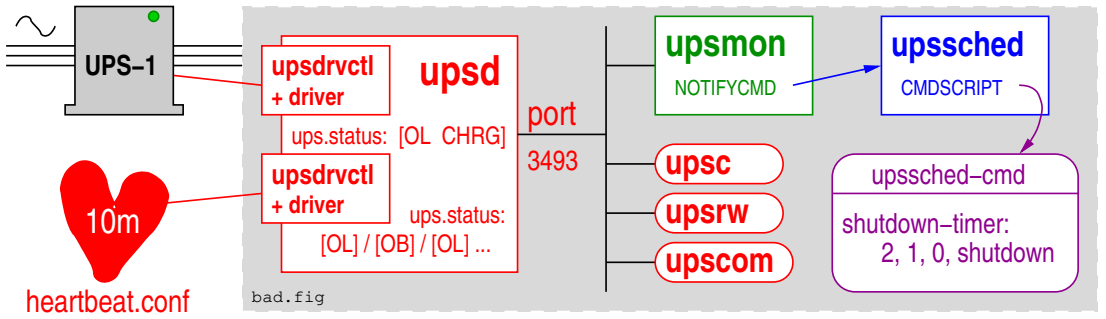


Figure 46: Workstation with timed shutdown.

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 20.
2. The upsd UPS declarations `ups.conf`: See chapter 7.1.
3. Configuration file `heartbeat.conf` which defines the dummy UPS providing the heartbeat. See chapter 7.2.
4. The upsd daemon access control `upsd.conf`: See chapter 7.3.
5. The upsd user declarations `upsd.users`: See chapter 7.4.
6. The upsmon daemon configuration: `upsmon.conf`. See chapter 7.5.
7. The upssched configuration: `upssched.conf`. See chapter 7.6.
8. The upssched-cmd script: see chapter 7.7.

9. The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix 21.

7.1 Configuration file ups.conf for workstation with timed shutdown

```
335 # ups.conf, timed shutdown
336 [UPS-1]
337    driver = usbhid-ups
338    port = auto
339    desc = "Eaton ECO 1600"
340    offdelay = 60
341    ondelay = 70
342    lowbatt = 33
343
344 [heartbeat]
345    driver = dummy-ups
346    port = heartbeat.conf
347    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 336 and 344 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 337 and 345 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 338 and 346 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 339 and 347 provide descriptive texts for the UPS.

For a detailed discussion of `offdelay` and `ondelay` on lines 340-341, see chapter 2.7.

Additional line 342 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[6] will restore the default, so if you want a permanent change you must change the default. See also chapter 2.10.

[6]List needed

7.2 Configuration file heartbeat.conf for workstation with timed shutdown

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

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

Figure 48: Configuration file heartbeat.conf 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.conf 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 349 and 351 flip the ups.status value between [OL] and [OB].

Lines 350 and 352 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

```
353 # upsd.conf
354 LISTEN 127.0.0.1 3493
355 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 37-38.

Line 354 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 355.

7.4 Configuration file upsd.users with timed shutdown

```
356 # upsd.users
357 [upsmaster]
358    password = sekret
359    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 40-42. For good security, ensure that only users upsd/nut and root can read and write this file.

Line 357 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 358 provides the password. You may prefer something better than "sekret".

Line 359 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.

```
360 # upsmon.conf
361 MONITOR UPS-1@localhost     1 upsmaster sekret master
362 MONITOR heartbeat@localhost 0 upsmaster sekret master
363 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 361

- The UPS name UPS-1 must correspond to that declared in ups.conf line 336.
- 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 40.
- sekret is the password declared in upsd.users line 41.
- master means this system will shutdown last, allowing any slaves time to shutdown first. There are no slaves in this simple configuration.

Line 362 declares that upsmon is also to monitor the heartbeat.

On line 363, `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.

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

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

Line 364 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 365 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 366, `POLLFREQ`, declares that the upsmon daemon will poll upsd every 5 seconds.

Line 367, `POLLFREQALERT`, declares that the upsmon daemon will poll upsd every 5 seconds while the UPS in on battery.

Line 368, `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 369, `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 370-379 assign a text message to each NOTIFY event. Within each message, the marker `%s` is replaced by the name of the UPS which has produced this event. upsmon passes this message to program `wall` to notify the system administrator of the event. You can change the default messages to something else if you like. The format is `NOTIFYMSG event "`*`message`*`"` where `%s` is replaced with the identifier of the UPS in question. Note that program `wall` has not been internationalized and

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

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

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 365.

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

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

Lines 380-389 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 380-381 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 365.

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 390 say that this happens every RBWARNTIME = 43200 seconds (12 hours).

Line 391: 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.

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

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

Line 392: 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 364. If you need to let your users do something in between those events, increase this number. Remember, at this point your UPS battery is almost depleted, so don't make this too big. Alternatively, you can set this very low so you don't wait around when it's time to shut down. Some UPS's don't give much warning for low battery and will require a value of 0 here for a safe shutdown.

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

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.

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

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

On line 394 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 395 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 395 the Debian distribution uses `/var/run/nut/upssched.pipe`.

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

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

Daemon upsmon requires the `LOCKFN` declaration on line 396 to avoid race conditions. The directory should be the same as `PIPEFN`.

Line 398 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 398 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 399 and 400 do the same thing for the 65 second timer `one-minute-warning-timer` and the 125 second timer `shutdown-timer`.

Line 402 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 403 and 404 do the same thing for the 65 second timer "`one-minute-warning-timer`" and the 125 second timer "`shutdown-timer`".

Line 405 command `EXECUTE` says that upssched is to call the user script immediately with the argument "`ups-back-on-line`".

On line 407, 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 408, 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

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

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

423 UPS="UPS-1"
424 STATUS=$( upsc $UPS ups.status )
425 CHARGE=$( upsc $UPS battery.charge )
426 CHMSG="[$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 421 and 422, change the e-mail addresses to something that works for you.

Lines 423-426 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 428-434 introduce the `heartbeat-failure-timer` case into the case statement. Line 429 specifies a message to be logged with the current UPS status as defined on lines 423-426.

Lines 430-432 compose a message to the sysadmin which is sent on line 433. The message includes the current state of those NUT kernel processes which are operational.

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

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

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

7.7.1 The timed shutdown

The cases at lines 435 and 437 specify warnings to be notified to the users when the `two-minute-warning-timer` and `one-minute-warning-timer` complete.

Beginning at line 439 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 441 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 442 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 23 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 381 upsmon calls upssched, specified by `NOTIFYCMD` on line 365. 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 398-400. 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 394 with the timer name as argument. In the script, this matches the case on line 435 which defines a suitable warning message in Bash variable `MSG`. Line 447 logs this message and line 448 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 437 which defines a stronger warning message in Bash variable `MSG`. Line 447 logs this message and line 448 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 439 which defines an ultimate warning message in Bash variable `MSG`, and then calls upsmon for a system shutdown. Line 447 logs message `MSG` and line 448 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 392.
10. upsmon creates `POWERDOWN` flag specified on line 369.
11. upsmon calls the `SHUTDOWNCMD` specified on line 364.

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 340.
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.

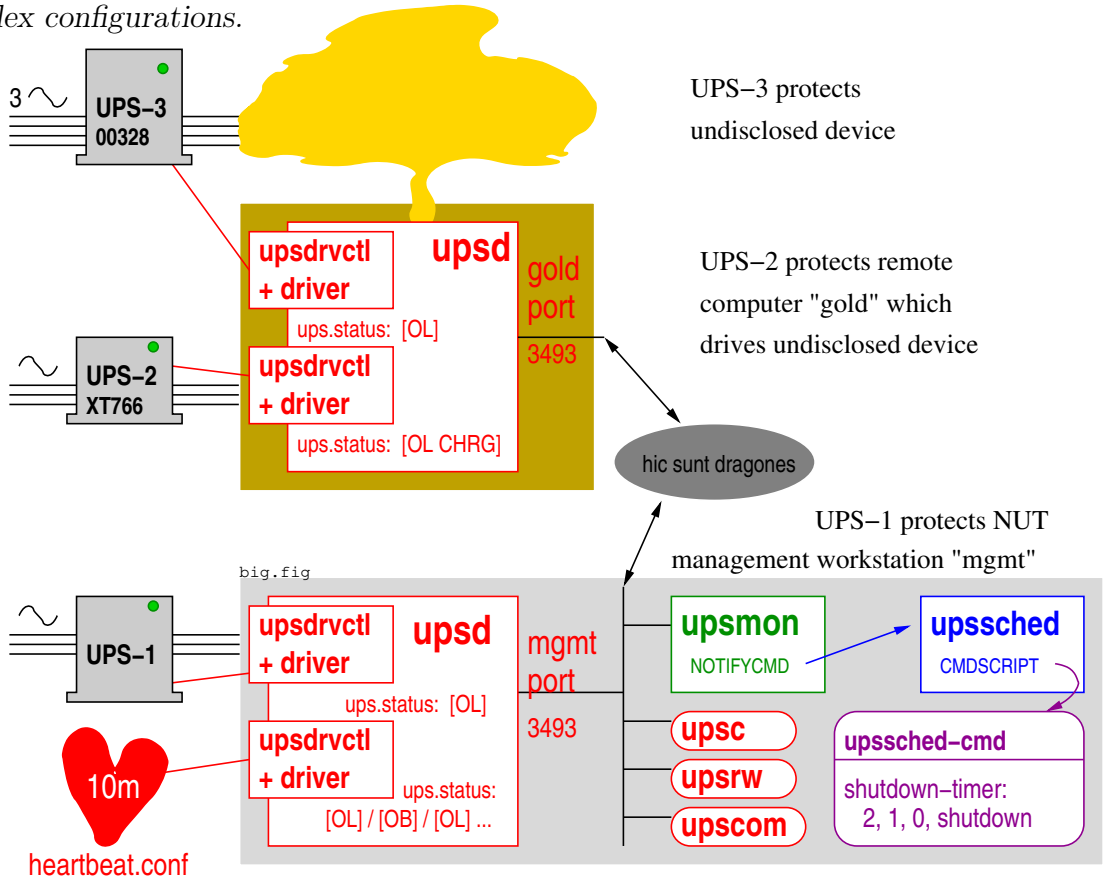


Figure 59: Workstation with additional equipment.

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 20.

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 21. 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 20.
7. mgmt: The upsd UPS declarations `ups.conf`: See chapter 8.2.
8. mgmt: The upsd heartbeat declaration `heartbeat.conf`: See chapter 8.2.
9. mgmt: The upsd daemon access control `upsd.conf`: See chapter 8.3.
10. mgmt: The upsd user declarations `upsd.users`: See chapter 8.4.
11. mgmt: The upsmon daemon configuration `upsmon.conf`: See chapter 8.5.
12. mgmt: The upssched configuration `upssched.conf`: See chapter 8.6.
13. mgmt: The upssched-cmd script: See chapter 8.7.
14. mgmt: The delayed UPS shutdown script. Since this file is common to all configurations, it is discussed separately in appendix 21.

8.1 Configuration files `nut.conf`

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

gold

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

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

mgmt

```
451 # nut.conf  -- mgmt --
452 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 450 and the init/systemd startup of the NUT software to ensure that only the upsd and driver daemons get started. See appendix 20. See also `man nut.conf`.

8.2 Configuration files ups.conf and heartbeat.conf

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

gold

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

Figure 62: File ups.conf for gold.
fig:upsconf.gold

mgmt

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

Figure 63: File ups.conf for mgmt.
fig:upsconf.mgmt

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

Figure 64: heartbeat.conf for mgmt.
fig:heartbeatconf.mgmt

gold: On lines 454-463 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 464 and 455 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 458-459 and 467-468 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 461 and 470. See `man usbhid-ups`.

mgmt: On lines 472-477 we offer a specimen definition for `UPS-1` and on lines 485-488 we propose the dummy UPS "heartbeat" discussed in chapter 6. The heartbeat requires the definition file heartbeat.conf, lines 485-488, to be placed in the same directory as ups.conf.

8.3 Configuration files upsd.conf

gold

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

Figure 65: File upsd.conf for gold.
fig:upsdconf.gold

mgmt

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

Figure 66: File upsd.conf for mgmt.
fig:upsdconf.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 490 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 490 and 491 for your own needs.

mgmt: Line 493 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 491 and 494.

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

8.4 Configuration files upsd.users

gold

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

Figure 67: File upsd.users for gold.

mgmt

```
499 # upsd.users   -- mgmt --
500 [upsmaster]
501    password = sekret
502    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 496 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 497 provides the password. You may prefer something better than "sekret".

Line 498 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 500 declares the "user name" of the system administrator who has write access to `UPS-1` and to the heartbeat managed by upsd.

Line 501 provides another uberl33t password.

Line 502 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.

```
503 # upsmon.conf  -- mgmt --
504 MONITOR UPS-3@gold          0 upsmaster sekret master
505 MONITOR UPS-2@gold          0 upsmaster sekret master
506 MONITOR UPS-1@localhost     1 upsmaster sekret master
507 MONITOR heartbeat@localhost 0 upsmaster sekret master
508 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 504 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 468.
- 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 496.
- `sekret` is the l33t password declared in `upsd.users` line 497.
- `master` means this system will shutdown last, allowing any slaves time to shutdown first. There are no slaves on gold.

Line 505 specifies that upsmon on mgmt will also monitor `UPS-2` which supplies the gold computer.

Line 506 specifies that upsmon on mgmt will monitor UPS-1 which supplies power to mgmt itself. Note the "power value" of 1.

Line 507 declares that upsmon is also to monitor the heartbeat.

On line 508, 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.

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

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

Line 509 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 510 says which program is to be invoked when upsmon detects a NOTIFY event flagged as EXEC.

Line 511, POLLFREQ, declares that the upsmon daemon will poll upsd in gold and in mgmt every 5 seconds.

Line 512, POLLFREQALERT, declares that the upsmon daemon will poll the upsd daemons every 5 seconds while any UPS in on battery.

Line 513, 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 514, 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.

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

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

Lines 515-524 assign a text message to each NOTIFY event. Within each message, the marker `%s` is replaced by the name of the UPS which has produced this event. On mgmt upsmon passes this message to program `wall` to notify the system administrator of the event. You can change the default messages to something else if you like. The format is `NOTIFYMSG event "`*`message`*`"` where `%s` is replaced with the identifier of the UPS in question. Note that program `wall` has not been internationalized and does not support accented letters or non latin characters. When the corresponding `NOTIFYFLAG` contains the symbol `EXEC`, upsmon also passes the message to the program specified by `NOTIFYCMD` on line 510.

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

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

Lines 525-534 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 525-526 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 510.

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.*

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

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

When a UPS says that it needs to have its battery replaced, upsmon will generate a [REPLBATT] NOTIFY event. Line 535 say that this happens every RBWARNTIME = 43200 seconds (12 hours).

Line 536: 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 537: 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 364. If you need to let your users do something in between those events, increase this number. Remember, at this point your UPS battery is almost depleted, so don't make this too big. Alternatively, you can set this very low so you don't wait around when it's time to shut down. Some UPS's don't give much warning for low battery and will require a value of 0 here for a safe shutdown.

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

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 539 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 540 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 540 the Debian distribution uses /var/run/nut/upssched.pipe.

Daemon upsmon requires the LOCKFN declaration on line 541 to avoid race conditions. The directory should be the same as PIPEFN.

8.6.1 UPS-3 on gold

Lines 543 and 544 say what is to be done by upssched for a NOTIFY event [ONBATT] due to UPS-3 on gold. On line 543 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,

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

Figure 74: Configuration file upssched.conf for mgmt .

upssched calls the user script specified by CMDSCRIPT with argument "UPS-3-two-minute-warning-timer". Line 544 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 545-547 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 547 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 549 and 550 define the timers which start when upssched receives a NOTIFY event [ONBATT], and lines 551 and 552 cancel those timers when hopefully upssched receives NOTIFY event [ONLINE].

Line 553 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 555 and 556 define the timers which start when upssched receives a NOTIFY event [ONBATT], and lines 557 and 558 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 559 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 561, 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 562, 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

```
563 #!/bin/bash -u
564 # upssched-cmd  -- mgmt --
565 logger -i -t upssched-cmd Calling upssched-cmd $1
566
567 # Send emails to/from these addresses
568 EMAIL_TO="sysadmin@example.com"
569 EMAIL_FROM="upssched-cmd@${HOSTNAME:-nut}.example.com"
570
571 function make-STCH {
572    STCH="[$( upsc $1 ups.status )]:$( upsc $1 battery.charge )%"}
573 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 568 and 569, change the e-mail addresses to something that works for you.

Lines 571-572 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 573 covering all the possible parameter values (timer names) that the user script may expect.

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

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

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

In figure 76, lines 574-580 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 578 will shut down the undisclosed device attached to gold. This code might for example be based on the `saltstack` remote management tools.

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

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

In figure 77, lines 582-588 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 586 will shut down the workstation.

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

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

In figure 78, lines 589-595 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 591, 592 and 593 prepare a message for the sysadmin in Bash variables `MSG1`, `MSG2` and `MSG3`. Lines 594-595 e-mail the message to the sysadmin. The message includes the current state of those NUT kernel processes which are operational.

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

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

In figure 79, lines 596-597 cover any unexpected parameter values, and lines 599-600 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 Encrypted connections – *Deprecated – to be removed*

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

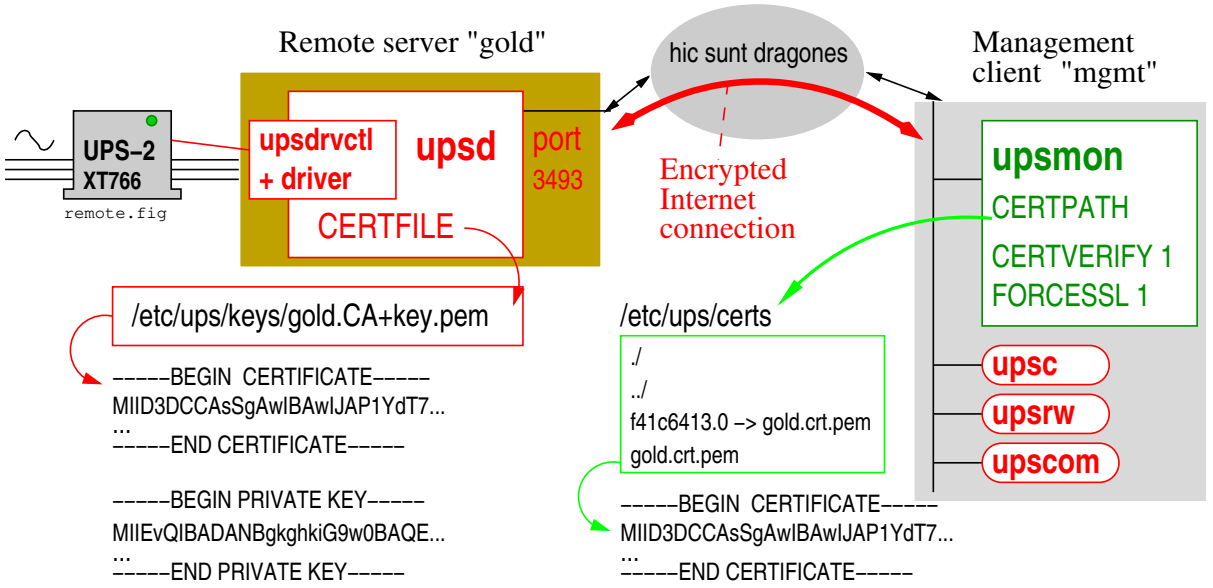


Figure 80: Encrypted connection to remote server using OpenSSL.

9.1 Waiting for NUT release 2.7.5

See NUT development Issues openssl 1.1 support #429, Add support for openssl-1.1.0 #504. and ./configure --with-openssl fails with OpenSSL 1.1, SSL_library_init now a macro #571 which are still outstanding and will not be fixed until NUT version 2.7.5 at the earliest.

Meanwhile this chapter contains my raw notes on the subject: they were obtained using a custom version of NUT rebuilt with OpenSSL 1.1. *Rebuilding NUT is beyond the scope of this tutorial.* They have **not** been tested.

9.2 Warning for Debian users

This chapter uses the OpenSSL libraries for SSL/TLS support. The function is provided by NUT but the Debian distribution has chosen to exclude OpenSSL saying "The OpenSSL licence taints the GNU GPL". This chapter has been developed using OpenSUSE 42.3 which includes OpenSSL support.

9.3 Introduction

SSL and the TLS that has replaced SSL are a quagmire of technical terms many of which are out-of-date, confusing or incorrectly used. The OpenSSL project has produced a Swiss Army Knife[7] of utilities which are the best known tools for work in this area. Anyone venturing into this mess has to do a lot of reading. Here is a very short list.

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

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

Certificate The public key used by clients to communicate with the server, possibly with additional information.

Certificate Authority (CA) Commercial businesses and others who want their customers to feel safe using their sites have their SSL certificates verified by a Certificate Authority (CA). You apply with a CSR, pay and receive a copy of your certificate linked to a trusted root certificate, for some meaning of "trust". Where does NUT stand? We are our own Certificate Authority and the certificate we create is itself the root certificate. We trust ourselves. In a closed industrial context where few people have access to the systems, this provides better security than the commercial offerings used on the web.

PEM PEM is an encoding [8] format for a certificate which is already ASN1 encoded and which allows it to be included in "ascii" base 64 files. If you are curious, the three letters PEM stand for Privacy-enhanced Electronic Mail. We use file type `.crt.pem` for these certificate files, but you will also find such certificates with just the `pem` extension. In our case the certificate is self-signed. It looks like this:

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

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

```
-----BEGIN CERTIFICATE-----
MIID3DCCAsSgAwIBAgIJAP1YdT7NA27mMA0GCSqGSIb3DQEBCwUAMIGCMQswCQYD
...
-----END CERTIFICATE-----
```

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

KEY The private key. We use file extension `.key.pem` for PEM-encoded keys which look like this:

```
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCw3bkc3NlA+2JH
...
-----END PRIVATE KEY-----
```

If the file also contains the Certificate Authority certificate (public key), we use the file extension `.CA+key.pem` .

9.3.1 Additional configuration files

The following configuration files are needed for encrypted communication between a remote NUT server and management client.

- In the remote server, code name gold :

  1. gold : The upsd daemon access control `upsd.conf` needs the private key generated by OpenSSL. The `CERTFILE`[9] declaration declares the file containing this private key in PEM format. Normally it is public keys that are refered to as "certificates". See chapter 9.6.
  2. gold : New directory `/etc/ups/keys` will hold the private key file. Debian users might use directory `/etc/nut/keys`.

- In each management client, code name mgmt :

  1. mgmt : The upsmon daemon configuration `upsmon.conf` needs the additional `CERTPATH`, `CERTVERIFY` and `FORCESSL` declarations: See chapter 9.7. `CERTPATH` points to a directory rather than a single file. This directory contains CA certificates in PEM format, used to verify the server certificate presented by the upsd server. The files each contain one CA certificate. The files are looked up by the CA subject name hash value, which must hence be available. See `man upsmon.conf`.

[9] The name "`CERTFILE`" is a poor choice since it is a private key not a public key. A name such as "`KEYFILE`" would have been better.

2. mgmt : New directory `/etc/ups/certs` will hold the certificate (public key) files. Debian users might use directory `/etc/nut/certs`.

9.4 Sniffing port 3493

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

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

1. In the server, gold , or in the management client mgmt , run the command `tcpdump -A port nut` as root.
2. In the management client mgmt , stop upsmon, and then restart it with the command `systemctl start nut-monitor.service`.
3. `tcpdump` will display the trace shown in figure 81 which has been edited to make it easier to read. Line 605 shows the client mgmt attempting to begin an encrypted session which is refused by server gold on line 607. Line 611 shows the password transmitted in clear text. *Let this be a warning to you.*

   Lines 617-620: Client mgmt then makes a plain text request every 5 seconds for the status of `UPS-3` which the server gold then answers in plain text.

```
601 listening on wlan0, link-type EN10MB (Ethernet), capture size 262144 bytes
602 IP mgmt.33656 > gold.nut:
603 IP gold.nut > mgmt.33656:
604 IP mgmt.33656 > gold.nut:
605 IP mgmt.33656 > gold.nut: STARTTLS
606 IP gold.nut > mgmt.33656:
607 IP gold.nut > mgmt.33656: ERR FEATURE-NOT-CONFIGURED
608 IP mgmt.33656 > gold.nut:
609 IP mgmt.33656 > gold.nut: USERNAME upsmaster
610 IP gold.nut > mgmt.33656: OK
611 IP mgmt.33656 > gold.nut: PASSWORD sekret
612 IP gold.nut > mgmt.33656: OK
613 IP mgmt.33656 > gold.nut: LOGIN UPS-3
614 IP gold.nut > mgmt.33656: OK
615 IP mgmt.33656 > gold.nut: MASTER UPS-3
616 IP gold.nut > mgmt.33656: OK MASTER-GRANTED
617 IP mgmt.33656 > gold.nut: GET VAR UPS-3 ups.status
618 IP gold.nut > mgmt.33656: VAR UPS-3 ups.status "OL"
619 IP mgmt.33658 > gold.nut:
620 IP mgmt.33656 > gold.nut: GET VAR UPS-3 ups.status
621 IP gold.nut > mgmt.33656: VAR UPS-3 ups.status "OL"
```

Figure 81: tcpdump of `systemctl start nut-monitor.service` without encryption.

9.5 Creating the SSL keys with OpenSSL

1. On gold, create a directory associated with NUT in which to build the keys. Since we use openSUSE, we will create a `keys` subdirectory of the server configuration directory `/etc/ups`. Debian sysadmins use `/etc/nut`. See table 126 for a list of possible directories. See lines 623-624. Note the ownership of directory `keys`.

2. On line 625, we `cd` into the `keys` subdirectory of the server configuration, and proceed to build a self-signed certificate. We are our own Certificate Authority (CA). On line 626, the command `openssl req` instructs the OpenSSL tool `req` to manage Certificate Signing Requests (CSR). The remaining options are specific to CSR management.

   On line 627, option `-newkey rsa:2048` calls for a new private key of length 2048 bits. Option `-nodes` says that there is no pass-phrase to encrypt the output key. The absence of a pass-phrase makes it possible to start the service automatically without having to type the pass-phrase. Option `-keyout NUT.key.pem` says where the private key is to be stored.

   On line 628, option `-x509` calls for `openssl req` to output an X509 structure instead of a certificate signing request (CSR). This is equivalent to saying "output a self-signed certificate". Option `-days 3660` says that the certificate is to be valid for 10 years. Option `-out NUT.CAcrt.pem` says into which file the certificate goes. The letters "`CA`" are a reminder that

```
622 root@gold ~ # cd /etc/ups
623 root@gold /etc/ups # mkdir keys
624 root@gold /etc/ups # chown root:nut keys
625 root@gold /etc/ups # cd keys
626 root@gold /etc/ups/keys # openssl req \
627 >        -newkey rsa:2048 -nodes -keyout NUT.key.pem \
628 >        -x509 -days 3660 -out NUT.CAcrt.pem
629 Generating a 2048 bit RSA private key
630 ..........+++
631 ..............................+++
632 writing new private key to 'NUT.key.pem'
633 -----
634 You are about to be asked to enter information that will be incorporated
635 into your certificate request.
636 What you are about to enter is what is called a Distinguished Name or a DN.
637 There are quite a few fields but you can leave some blank
638 For some fields there will be a default value,
639 If you enter '.', the field will be left blank.
640 -----
641 Country Name (2 letter code) [AU]:FR
642 State or Province Name (full name) [Some-State]:.
643 Locality Name (eg, city) []:.
644 Organization Name (eg, company) [Internet Widgits Pty Ltd]:Roger Price
645 Organizational Unit Name (eg, section) []:Network UPS Tools (NUT)
646 Common Name (e.g. server FQDN or YOUR name) []:gold.example.com
647 Email Address []:sysadmin@example.com
```

Figure 82: Call `openssl req` to create the self-signed certificate.

this is the Certifying Authority public key.

3. The openssl command on line 626 produces the two files in directory `/etc/ups/keys` shown on lines 649 and 650. Let's look at the contents of these two files:

9.5.1 Create unique name for certificate using OpenSSL

Later, when installing the certificate (public key) on mgmt, we will need a unique name for this file. We create this name now on gold using the `openssl x509` tool.

The file name will be `f41c6413.0` which will be used on line 683.

9.6 Install NUT server keys on gold

The upsd server on gold requires that the certificate and the private key generated by openssl be in one single file. This file must have ownership and permissions which prevent public access, but

```
root@gold /etc/ups/keys # ls -alF
-rw-r--r-- 1 root root 1399 Jun 30 16:35 NUT.CAcrt.pem
-rw------- 1 root root 1704 Jun 30 16:29 NUT.key.pem
root@gold /etc/ups/keys # grep -A1 "\-" NUT.CAcrt.pem
-----BEGIN CERTIFICATE-----
MIID6TCCAtGgAwIBAgIUDWeXm6QFobVRzpb+1E2sSnBQDhEwDQYJKoZIhvcNAQEL
--
-----END CERTIFICATE-----
root@gold /etc/ups/keys # grep -A1 "\-" NUT.key.pem
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQC5Bn7udfNGVSON
--
-----END PRIVATE KEY-----
```

Figure 83: The contents of the two files produced by `openssl req`.

```
root@gold /etc/ups/keys # openssl x509 -hash -noout -in NUT.CAcrt.pem
f41c6413
```

Figure 84: Create unique name for certificate file.

just allow upsd to read the file. We proceed as follows:

```
root@gold /etc/ups/keys # cat NUT.CAcrt.pem NUT.key.pem > gold.CA+key.pem
root@gold /etc/ups/keys # chown root:upsd gold.CA+key.pem
root@gold /etc/ups/keys # chmod 0640 gold.CA+key.pem
root@gold /etc/ups/keys # ls -alF gold.CA+key.pem
-rw-r----- 1 root upsd 3103 Jul  1 08:56 gold.CA+key.pem
```

Figure 85: The combined file required by upsd on gold.

On line 663 `NUT.CAcrt.pem` must come before `NUT.key.pem`. On line 664, Debian sysadmins would prefer `chown root:nut`... Line 671 extends the file `upsd.conf` on gold to include a `CERTFILE` declaration which points to `gold.CA+key.pem` created on line 663.

```
# upsd.conf
LISTEN 127.0.0.1 3493
LISTEN ::1 3493
CERTFILE /etc/ups/keys/gold.CA+key.pem      OpenSUSE
# CERTFILE /etc/nut/keys/gold.CA+key.pem     Debian
```

Figure 86: `CERTFILE` declaration to be added to `upsd.conf` on gold.

9.7 Install NUT management client keys on mgmt

1. On mgmt, create a directory associated with NUT in which to store the certificate (public key). Since we use openSUSE, we will create a `certs` subdirectory of the configuration directory `/etc/ups`. Debian sysadmins use `/etc/nut`. See table 126 for a list of possible directories. See lines 674-675. Note the ownership of directory `certs`. On line 675 Debian sysadmins would prefer `chown root:nut`...

```
673 root@mgmt ~ # cd /etc/ups
674 root@mgmt /etc/ups # mkdir certs
675 root@mgmt /etc/ups # chown upsd:root certs
676 root@mgmt /etc/ups # cd certs
677 root@mgmt /etc/ups/certs # sftp gold:/etc/ups/keys/NUT.CAcrt.pem gold.crt.pem
678 root@gold's password:
679 Connected to gold.
680 Fetching /etc/ups/keys/NUT.CAcrt.pem to gold.crt.pem
681 /etc/ups/keys/NUT.CAcrt.pem          100% 1399   183.6KB/s   00:00
682 root@mgmt /etc/ups/certs # chown upsd:root gold.crt.pem
683 root@mgmt /etc/ups/certs # ln -s gold.crt.pem f41c6413.0
684 root@mgmt /etc/ups/certs # ls -alF
685 lrwxrwxrwx 1 root root    9 Jul  3 16:56 f41c6413.0 -> gold.crt.pem
686 -rw-r--r-- 1 upsd root 1399 Jul  3 15:17 gold.crt.pem
```

Figure 87: Copy certificate to mgmt and rename file.

2. Line 677: copy the certificate (public key) from gold to mgmt. Line 682 corrects the ownership for OpenSUSE. A Debian sysadmin would prefer `chown nut:root`...
3. Line 683 links the unique name `f41c6413.0` generated on line 661 to the file `gold.cert.pem`.
4. Add a `CERTPATH` declaration to `upsmon.conf`. Here is figure 69 modified with additional `CERTPATH`, `CERTVERIFY` and `FORCESSL` declarations on lines 692-694.

```
687 # upsmon.conf  -- mgmt --
688 MONITOR UPS-3@gold           0 upsmaster sekret master
689 MONITOR UPS-2@gold           0 upsmaster sekret master
690 MONITOR UPS-1@localhost      1 upsmaster sekret master
691 MONITOR heartbeat@localhost 0 upsmaster sekret master
692 CERTPATH /etc/ups/certs
693 CERTVERIFY 1
694 FORCESSL 1
695 MINSUPPLIES 1
```

Figure 88: Configuration file `upsmon.conf` for mgmt, with `CERTFILE`.

9.8 Testing the TLS setup

On gold restart upsd with command `systemctl restart nut-server.service` and then command `systemctl status nut-server.service` . The report should show

```
696 nut-server.service - Network UPS Tools - power devices information server
697    Loaded: loaded (/usr/lib/systemd/system/nut-server.service; enabled;..)
698    Active: active (running) since Sat 2018-07-07 11:01:40 CEST; 51min ago
699   Process: 2923 ExecStart=/usr/sbin/upsd (code=exited, status=0/SUCCESS)
700  Main PID: 2926 (upsd)
701     Tasks: 1 (limit: 512)
702    CGroup: /system.slice/nut-server.service
703            \_2926 /usr/sbin/upsd
704
705 ... upsd[2923]: listening on 0.0.0.0 port 3493
706 ... upsd[2923]: Connected to UPS [UPS-2]: usbhid-ups-UPS-2
707 ... upsd[2923]: Connected to UPS [UPS-3]: usbhid-ups-UPS-3
708 ... upsd[2926]: Startup successful
709 ... systemd[1]: Started Network UPS Tools - power device information server
710 ... upsd[2926]: User upsmaster@gold logged into UPS [UPS-2] (SSL)
711 ... upsd[2926]: User upsmaster@gold logged into UPS [UPS-3] (SSL)
```

Figure 89: Restarting upsd on gold with SSL/TLS enabled.

On mgmt restart NUT with command `systemctl restart nut-monitor.service` and then command `systemctl status nut-monitor.service` . The report should show

Lines 723-726 show that the upsmon connections are SSL/TLS encrypted. Line 729 shows the heartbeat in action.

```
nut-monitor.service - Network UPS Tools - power device monitor and shutdown
   Loaded: loaded (/usr/lib/systemd/system/nut-monitor.service; enabled;..)
   Active: active (running) since Sat 2018-07-07 11:01:40 CEST; 51min ago
  Process: 2927 ExecStart=/usr/sbin/upsmon (code=exited, status=0/SUCCESS)
 Main PID: 2931 (upsmon)
    Tasks: 3 (limit: 512)
   CGroup: /system.slice/nut-monitor.service
           |-2930 /usr/sbin/upsmon
           |-2931 /usr/sbin/upsmon
           \_3591 /usr/sbin/upssched UPS heartbeat@localhost: On battery

... upsmon[2931]: Connected to gold in SSL
... upsmon[2931]: Connected to gold in SSL
... upsmon[2931]: Connected to localhost in SSL
... upsmon[2931]: Connected to localhost in SSL
... upssched[3591]: Timer daemon started
... upssched[3591]: New timer: heartbeat-failure-timer (1320 seconds)
... upssched[3591]: Cancelling timer: heartbeat-failure-timer
... upssched[3591]: New timer: heartbeat-failure-timer (1320 seconds)
```

Figure 90: Restarting upsmon on mgmt with SSL/TLS enabled.

9.9 What can Debian users do?

Debian users have a choice:

1. Rebuild NUT with the `./configure` option `--with-openssl` *Rebuilding NUT is beyond the scope of this tutorial. See NUT issue 571.*
2. Use the NSS support which _is_ included in the Debian NUT package. See Mozilla Network Security Services (NSS). *See also NUT issue 572.*

9.9.1 Debian: Create NSS database on gold

The NSS instructions given in the Network UPS Tools User Manual, chapter 9, Notes on securing NUT correspond to earlier versions of NSS. We choose to use the current version and to base the setup on key creation done with OpenSSL, so the instructions here differ from those in the NUT User Manual.

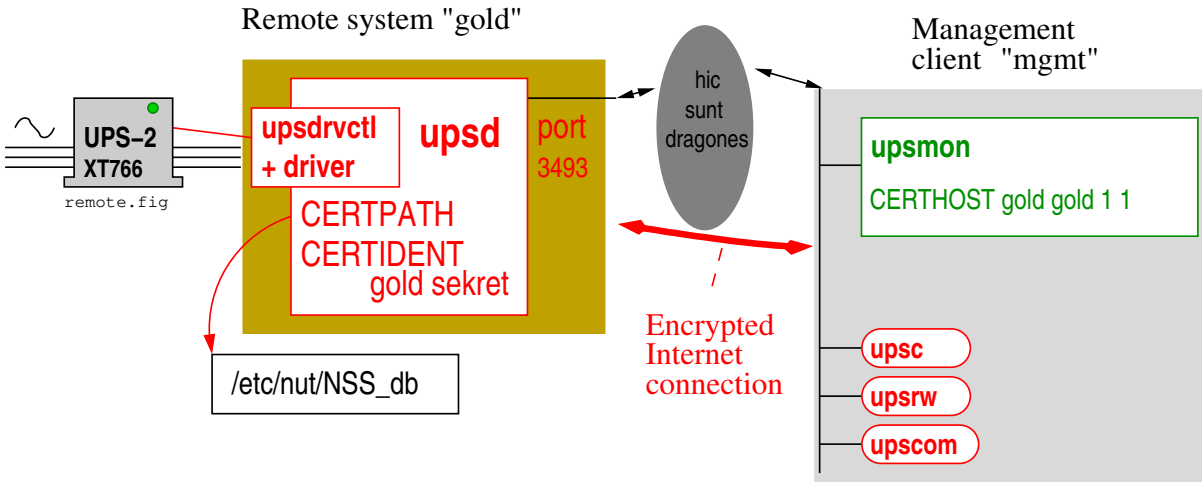


Figure 91: Encrypted connection to remote server using NSS.

There are two different forms for the NSS database: the legacy databases (`cert8.db`, `key3.db`, and `secmod.db`) and new SQLite databases (`cert9.db`, `key4.db`, and `pkcs11.txt`). These are identified by the prefixes `sql:` for the newer database and `dbm:` for the legacy database. NUT 2.7.4 does not provide a means of specifying the `sql:` prefix and does not support use of the newer `sql:` database.

We refer to these three databases collectively as the NSS database, which must be created on those Debian boxes which act as gold and mgmt, before certificates or keys can be imported and managed.

gold: Line 732: You will need package `libnss3-tools` for program `certutil` which creates the (initially empty) databases. Note the `dbm:` prefix which must be placed before all database references, and the weak approach to security shown by the `--empty-password` option.

Line 735 shows the ownership and permissions of the databases.

```
731 root@gold /etc/nut # mkdir NSS_db
732 root@gold /etc/nut # certutil -N -d dbm:NSS_db --empty-password
733 root@gold /etc/nut # chown -R root:nut NSS_db/
734 root@gold /etc/nut # chmod -R 640 NSS_db/
735 root@gold /etc/nut # ls -alF NSS_db/
736 drw-r----- 2 root nut  4096 Jul  8 12:40 .
737 drwxr-xr-x 5 root nut  4096 Jul  8 12:40 ../
738 -rw-r----- 1 root nut 65536 Jul  8 12:40 cert8.db
739 -rw-r----- 1 root nut 16384 Jul  8 12:40 key3.db
740 -rw-r----- 1 root nut 16384 Jul  8 12:40 secmod.txt
```

Figure 92: Creating the NSS databases on gold.

9.9.2 Debian: Add OpenSSL keys and certificates to NSS database on gold

The `certutil` tool is capable of many operations needed to create and manage certificates and keys, but we choose to use OpenSSL to build ours which we then import into the NSS database.

gold: Line 741: Use tool `openssl pkcs12` to export the private key `gold.key` to a PKCS#12 file `gold.p12` for NSS to import. Note the option `-name gold` which specifies the private key's nickname. On line line 744 tool `pk12util` imports the private key from file `gold.p12` into the NSS database.

```
741 root@gold /etc/nut # openssl pkcs12 -export -inkey ./keys/gold.key \
                         -in ./keys/gold.crt -out ./keys/gold.p12 -name gold
742 Enter Export Password: sekret
743 Verifying - Enter Export Password: sekret
744 root@gold /etc/nut # pk12util -i ./keys/gold.p12 -d dbm:NSS_db
745 Enter password for PKCS12 file: sekret
746 pk12util: PKCS12 IMPORT SUCCESSFUL
```

Figure 93: Import private key to NSS database on gold.

Now we have the private key in the NSS database, we also need the public key, i.e. the certificate.

Line 747: Use tool `openssl x509` to export the certificate (public key) in `gold.pem` to a DER format file `gold.der` for NSS to import. On line 748 tool `certutil -A` adds the certificate in file `gold.der` to the NSS database with option `-t "C,,"` declaring that the certicate is trusted for client authentification on an SSL server, option `-v 120` declaring that the certificate is valid for 10 years, and option `-n "gold"` specifying a nickname for the certificate.

Line 752 extends the file `upsd.conf` on gold to include a `CERTPATH` declaration which points to the NSS database. Line 753 identifies the certificate to be sent to clients and the password needed to decrypt the private key associated with the certificate, see line 745.

```
747 root@gold /etc/nut # openssl x509 -outform der \
                         -in ./keys/gold.pem -out ./keys/gold.der
748 root@gold /etc/nut # certutil -A -d dbm:NSS_db -t "C,," \
                          -v 120 -n "gold" -i ./keys/gold.der
```

Figure 94: Import certificate (public key) to NSS database on gold.

```
749 # upsd.conf -- gold -- for Debian
750 LISTEN 127.0.0.1 3493
751 LISTEN ::1 3493
752 CERTPATH /etc/nut/NSS_db
753 CERTIDENT "gold.example.com" sekret
```

Figure 95: NSS CERTPATH declaration for upsd.conf on gold.

9.9.3 Debian: Check and display NSS database on gold

We check the private key and certificate (public key) in the NSS database. See figure 96.

gold: Line 754: `certutil -V` checks the validity of a certificate, with the option `-n gold` giving the nickname of the key as defined on line 741, and option `-u V` declaring that the certificate is for use as an SSL server.

Line 756: `certutil -K` lists the contents of the key database. The key ID is `df7b...` with nickname `gold` as defined on line 741.

Line 759: `certutil -L` lists the certificates in the database. Specify nickname `gold` to get full detail for that certificate.

9.9.4 Debian: Create NSS database on mgmt

The process of creating the NSS database on mgmt is the same as on gold.

However file upsmon.conf requires specific attention.

9.9.5 Debian: Testing the NSS setup

On gold restart upsd with command `systemctl restart nut-server.service` and then command `systemctl status nut-server.service` . The report should show

On mgmt restart NUT with command `systemctl restart nut-monitor.service` and then command `systemctl status nut-monitor.service` . The report should show

```
root@gold /etc/nut # certutil -V -d dbm:NSS_db -n gold -u V
certutil: certificate is valid
root@gold /etc/nut # certutil -K -d dbm:NSS_db
certutil: Checking token "NSS Certificate DB" in slot
          "NSS User Private Key and Certificate Services"
< 0> rsa      df7b376946c8cfe59d74095dfc4b882d081b981b   gold
root@gold /etc/nut # certutil -L -d dbm:NSS_db -n gold
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            00:fd:58:75:3e:cd:03:6e:e6
        Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption
        Issuer: "E=sysadmin@rogerprice.org,CN=maria.rogerprice.org,
                 OU=IT operations,O=Roger Price,C=FR"
        Validity:
            Not Before: Sat Jun 30 14:35:24 2018
            Not After : Tue Jun 27 14:35:24 2028
...
```

Figure 96: Check and display certificate and private key on gold.

```
# upsmon.conf -- mgmt -- for Debian
MONITOR UPS-3@gold          0 upsmaster sekret master
MONITOR UPS-2@gold          0 upsmaster sekret master
MONITOR UPS-1@localhost     1 upsmaster sekret master
MONITOR heartbeat@localhost 0 upsmaster sekret master
CERTHOST gold gold.example.com 1 1
CERTVERIFY 1
FORCESSL 1
MINSUPPLIES 1
```

Figure 97: NSS `CERTHOST` declaration for `upsmon.conf` on mgmt.

Part 2

UPS monitoring using Python3 script and openSSL

Part 1 of this documentation discussed the way in which UPS activity reported by upsd can be monitored using the monitoring software provided with NUT 2.7.4. This part covers the use of Python3 scripts and openSSL to monitor the same UPS activity. Part 3 provides technical appendices.

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

This Part provides descriptions of Python3 scripts mkNUTcert.py, upsdTLS.py, UPSmon.py and mkUPSmonconf.py.

The scripts and their SHA1 check sums may be downloaded from http://rogerprice.org/NUT

10 mkNUTcert.py builds TLS certificates for NUT

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

10.1 Very Short Introduction to TLS Certificates

SSL and the TLS that has replaced SSL are a quagmire of technical terms many of which are out-of-date, confusing or incorrectly used. The OpenSSL project has produced a Swiss Army Knife[10] of utilities which are the best known tools for work in this area. Anyone venturing into this mess has to do a lot of reading. Here is a very short list.

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

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

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

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

Certificate Authority (CA) Commercial businesses and others who want their customers to feel safe using their sites have their TLS certificates verified by a Certificate Authority (CA). You apply with a CSR, pay and receive a copy of your certificate linked to a trusted root certificate, for some meaning of "trust". Where does NUT stand? We are our own Certificate Authority and the certificate we create is itself the root certificate. We trust ourselves. In a closed industrial context where few people have access to the systems, this provides better security than the commercial offerings used on the web.

Root certificate A Certifying Authority takes the private key and provides a certificate of authenticity known as a "root certificate". However in the commercial world intermediaries appear and get paid to add their certificates, thus forming a "chain of trust". NUT does not have such a chain. The root certificate is the only one. In NUT's self-signed world, the upsd server uses as private key a file which contains the private key and then the root certificate[11]. For the private key we use a file name of the form *`mybox`*`.cert.pem` where *mybox* is the name of the upsd server. The clients will use just the root certificate which contains the public key.

PEM PEM is an encoding [12] format for a certificate which is already ASN1 encoded and which allows it to be included in "ascii" base 64 files. If you are curious, the three letters PEM stand for Privacy-enhanced Electronic Mail. We use file type `.cert.pem` for these certificate files, but you will also find such certificates with just the `pem` extension.

[11] In that order

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

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

10.2 Overview of mkNUTcert.py

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

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

Figure 98: Command `mkNETcert.py --help`.

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

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

`-SAN` *<list of server names>* *See* `--subjectAltName`

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

`-C` *<ISO 3166 two letters>* *See* `--countryName`

`--countryName` *<ISO 3166 two letters>* Feel free to specify your 2 digit country code. The default is "`FR`".

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

`-OU` *<unit name>*, `--organisationUnitName` *<unit name>* The default value for the Organisation Unit name is "`mkNUTcert.py version 1.0`". Again, you probably don't have to change this.

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

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

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

`-s` *<filename>*, `--servercertfile` *<filename>* File path and name for the server's certificate. mkNUTcert.py tries to guess where to put things. Lucky users of Debian might see `/etc/nut/mkNUTcert/`*mybox*`.cert.pem` See table 126 for a list of possible directories.

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

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

The private key and public keys provided by mkNUTcert.py are in the form of PEM encoded certificates. The server's private key PEM encoding can be seen with command shown in figure 99:

```
786 $ grep -A1 -E "^---" /etc/ups/mkNUTcert/mybox.cert.pem
787 -----BEGIN PRIVATE KEY-----
788 MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQC2sJigLVujiJ0/
789 --
790 -----END PRIVATE KEY-----
791 -----BEGIN CERTIFICATE-----
792 MIIFhDCCA2ygAwIBAgIBATANBgkqhkiG9w0BAQ0FADBMMQswCQYDVQQGEwJGUjEa
793 --
794 -----END CERTIFICATE-----
```

Figure 99: The server's PEM encoded private key.

The monitor's public key contains only the CERTIFICATE part, not the PRIVATE KEY part. Details of the certificate can be seen with the command shown in figure 100:

Notes:

1. The certificate is a root certificate and there are no intermediate certificates. NUT acts as it's own certifying authority. For tightly controlled situations such as UPS management, this provides better security.
2. The certificate is self-signed. The issuer on line 801 is also the subject on line 805 as required by RFC 5280 para 4.1.2.4 last sentence.
3. The value "`Dec 31 23:59:59 9999 GMT`" on line 804 is defined by RFC 5280 para 4.1.2.5.

```
795 $ openssl x509 -text -noout -in /etc/nut/mkNUTcert/mybox.cert.pem
796 Certificate:
797   Data:
798     Version: 3 (0x2)
799     Serial Number: 1 (0x1)
800     Signature Algorithm: sha512WithRSAEncryption
801     Issuer: C = FR, O = Network UPS Tools, OU = mkNUTcert.py version 1.0
802     Validity
803       Not Before: Sep 27 14:19:02 2020 GMT
804       Not After : Dec 31 23:59:59 9999 GMT
805     Subject: C = FR, O = Network UPS Tools, OU = mkNUTcert.py version 1.0
806     Subject Public Key Info:
807       Public Key Algorithm: rsaEncryption
808         RSA Public-Key: (4096 bit)
809         Modulus:
810           00:b1:aa:dc:87:3c:ec:11:42:59:92:1d:5c:58:17:
811           ...
812         Exponent: 65537 (0x10001)
813     X509v3 extensions:
814       X509v3 Basic Constraints: critical
815        CA:TRUE
816       X509v3 Subject Alternative Name:
817         DNS:mybox, DNS:localhost, DNS:10.218.0.19, DNS:mybox.example.com
818       X509v3 Subject Key Identifier:
819         DA:39:A3:EE:5E:6B:4B:0D:32:55:BF:EF:95:60:18:90:AF:D8:07:09
820   Signature Algorithm: sha512WithRSAEncryption
821     3a:fb:9c:f9:a0:ea:a7:cf:85:af:fd:20:fb:62:5d:e5:07:3b:
822     ...
```

Figure 100: The self-signed certificate.

4. The public key begins on line 810.
5. There is no Authority Key Identifier which is obligatory for Web certificates. This omission is specific to self-signed certificates, see RFC 5280 para 4.2.1.1.

10.3 Running mkNUTcert.py

1. Before running the script, check the shebang `#!` in the first line. The default value is `#!/usr/bin/python3 -u` . Check that you have a sufficiently recent version of Python3 at that address. If your version is not sufficiently recent, you will receive an error message from mkNUTcert.py. How do I know if I have a sufficiently recent version of Python3? Try running the script. If it runs, you're ok. Otherwise you will need to upgrade your Python installation. See Annex 24.

2. Run command `mkNUTcert.py --help` to see the default values, particularly for options `--subjectAltName` , `--servercertfile` and `--monitorcertfile` .
3. When you run the command `mkNUTcert.py` you will be reminded of the proposed file paths and file names for the certificates. Enter "`yes`" to confirm and anything else to exit immediately.
4. Ensure that the private key is properly protected. Only root and the user designated to run upsd should have access to the key. No-one else.

11 Daemon upsdTLS.py

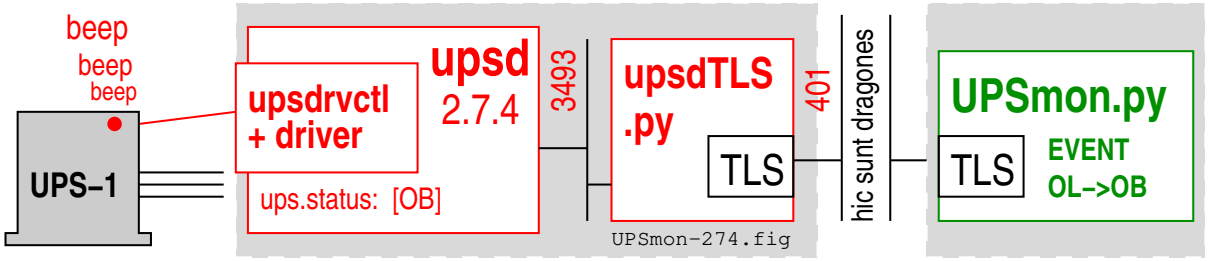


Figure 101: UPSmon.py with NUT 2.7.4 requires a TLS helper upsdTLS.py.

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

11.1 Overview of upsdTLS.py

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

```
$ upsdTLS.py --help
usage: upsdTLS.py  [-h] [--backlog <integer>] [-D] [--noTLS]
  [--servercertfile <file>] [--listen <IPv4_address> <port_number>]
  [--logfile <file>] [--maxconn <integer>] [--period <float>]
  [--upsdport <integer>] [--upsdtimeout <float>]
  [--montimeout <float>] [-u <user>] [-v]
```

Figure 102: Command `upsdTLS.py --help`

.

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

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

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

`-l <file>, --logfile <file>` The log file, with default `/var/log/NUT.log` . Progress and error messages and the stuff generated by option `-D` go into this file. See chapter 26.3 for an extension to `logrotate` to cover this file.

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

`-s <file>, --servercertfile <file>` The file path and file name of the server's private key. upsdTLS.py tries to guess where to put things. The default on Debian systems is `/etc/nut/mkNUTcert/mybox.cert.pem` . OpenSUSE sysadmins would probably use `/etc/ups/...` See table 126 for a list of possible directories.

`--listen <IPv4_address>, <port_number>` Listen to UPSmon.py on this interface and port, the default is `'127.0.0.1', 401`. We squat IANA ups/401. Setting a port number < 1024 requires starting the daemon as root.

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

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

`--upsdport <integer>` Relay incoming traffic from UPSmon to this upsd port, the default is 3493, the well known NUT port.

`--upsdtimeout <float>` Socket timeout for exchanges with upsd. The default is 0.8 seconds. Note that since upsdTLS.py is not protocol aware, it sometimes has to rely on timeouts to determine that a message exchange has completed.

`--montimeout <float>` Socket timeout for exchanges with UPSmon.py. The default is 1.8 seconds. As with upsd, upsdTLS.py sometimes has to rely on timeouts to determine that a message exchange has completed.

`-u <user>, --user <user>` After launch as root, run as this user. upsdTLS.py tries to guess the user. OpenSUSE admins would probably see `upsd`, whereas Debian admins would see `nut`. See table 126 for a list of possible users.

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

11.2 Running upsdTLS.py

The daemon upsdTLS.py usually starts with user root and forks to run as the same user as upsd.

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

Copy the service unit file `/usr/lib/systemd/system/nut-server.service` to `/etc/systemd/system/nut-py-server.service` and modify the new file shown in figure 103. Lines 831, 832 and 834-835 have been changed. The `PIDFile` declaration is there to help systemd find the daemon since upsdTLS.py does not keep the parent process running when it forks. Note that systemd service units in `/etc` take precedence over those in `/usr/lib`. See `man systemd.unit(5)`.

```
829 [Unit]
830 Description=Network UPS Tools - nut-server TLS support daemon
831 After=local-fs.target network.target nut-server.service
832 Before=nut-py-monitor.service

833 [Service]
834 ExecStart=/usr/sbin/upsdTLS.py
835 PIDFile=/var/run/upsdTLS.pid
836 Type=forking

837 [Install]
838 WantedBy=multi-user.target
```

Figure 103: systemd service unit `nut-py-server.service` for upsdTLS.py.

You may choose to place the upsdTLS.py script in directory `/usr/sbin` or make `/usr/sbin/upsdTLS.py` a link to wherever you put the Python script. After you have made the changes, you should run the command `systemctl daemon-reload` See `man systemctl(1)`. Before running upsdTLS.py the first time, you will need to run the command

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

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

`systemctl daemon-reload` to make any changes to the service unit available to systemd.

`systemctl enable nut-py-server.service` to make the daemon upsdTLS.py operational and "startable".

`systemctl start nut-py-server.service` to start upsdTLS.py. Note that this will not erase the log file. If you want to clear the log file then you need to do that yourself. See also chapter 26.3 for a discussion of log rotation.

`systemctl status nut-py-server.service` to see the current status of daemon upsdTLS.py.

`systemctl stop nut-py-server.service` to stop upsdTLS.py.

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

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

12 Python3 script UPSmon.py

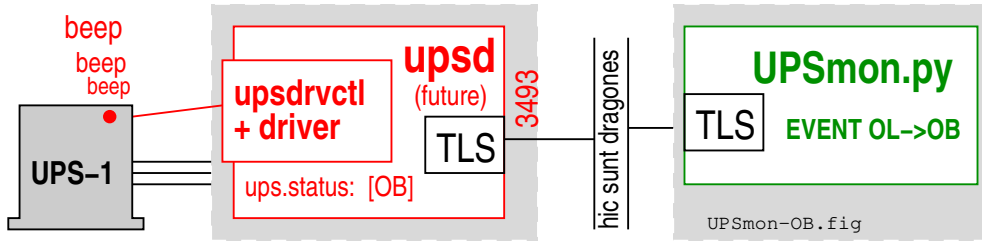


Figure 104: UPSmon.py requires TLS.

12.1 What is UPSmon.py ?

UPSmon.py is a Python3 script which replaces upsmon, upssched and upssched-cmd. The configuration files `upsmon.conf` and `upssched.conf` are replaced by a single configuration file `UPSmon.conf`. The current version of UPSmon.py is "experimental", intended for experiment and demonstration.

12.1.1 Principal differences between upsmon and UPSmon.py

The principal differences between NUT's upsmon and UPSmon.py are:

1. UPSmon.py is written in Python3 rather than K&R C. It is hoped that this use of a well known higher level language will encourage further experimentation. The script is in one single file rather than the many separate files used in NUT C code. Like the NUT C code, the script is **not** object oriented. To assist further development, the script provides 116 error and warning messages, and the `-D` and `-Y` debug options provide a detailed "walk-through" of the script's operations.
2. Unlike upsmon, UPSmon.py does not retain the parent process when forking to a non-privileged user. This improves security, but imples that the non-privileged user such as `nut` has `sudo` rights for programs `wall`, `notify-send` and `shutdown`.
3. UPSmon.py assumes that it will be managing a large number of physical and virtual UPS and other power supply units. The management may be of the type "master" or "slave" or simply as an observer with the master/slave shutdown decisions taken elsewhere.
4. The UPS units, real and virtual, are collected into groups. All UPS's must be in exactly one group.
5. All UPS's must be individually identified. Unlike NUT, there are no "wildcard" UPS's. Each UPS has a formal "fully qualified" name which is of the form *group*`:`*ups*`@`*host*`:`*port* , for example `HB:heartbeat@bigbox:3493` , although shortened forms are used where there is no ambiguity.

6. The configuration file `UPSmon.conf` is read by PLY, Python Lex and Yacc. This implies a slightly slower start-up than NUT but allows freer formats and many possibilities for future expansion.
7. The `upsmon.conf` declarations `DEADTIME`, `FINALDELAY`, `HOSTSYNC`, `NOCOMMWARNTIME` and `RBWARNTIME` are not needed in `UPSmon.conf` since they are timers which can be expressed directly if needed.
8. All communication between UPSmon.py and upsd is TLS encrypted. The version of OpenSSL used is too recent to be compatible with nut 2.7.4, so a front end for upsd called upsdTLS.py is provided to accept TLS encrypted messages from UPSmon.py and then relay that traffic to the local upsd. The options chosen for TLS call for the latest version with full checking of the certificates. Use of the earlier and now deprecated SSL is excluded.
9. UPSmon.py supports two loggers: the system log and a text based NUT-specific log.
10. UPSmon.py does not require a supplementary program such as upssched or a script such as upssched-cmd. The functions of those programs are available in UPSmon.py. NUT's upsmon provides three `NOTIFYFLAG` options: `SYSLOG`, `WALL` and `EXEC`, UPSmon.py replaces these with the more complete set of actions shown in figure 105.

Action	Effect
`STARTTIMER` *name value*	Start timer with the given name and value in seconds.
`CANCELTIMER` *name*	Cancel timer with the given name.
`EMAIL` `FROM` *text* `TO` *text* `SUBJECT` *text* `MESSAGE` *text*	Send email.
`WALL` *text*	Send text to local wall.
`NOTIFY` *text*	Place text on screens of all logged-in local accounts.
`PRINT` *text*	Send text to STDOUT.
`EPRINT` *text*	Send text to STDERR.
`NUTLOG` *text*	Send text to NUT-specific logger.
`SYSLOG` *text*	Send text to system logger.
`SETFSD` *name*	Send FSD to upsd for UPS *name*.
`SHUTDOWN` *option when*	Shutdown the system, e.g. with `/sbin/shutdown -h now`.
`DEBUG` *level*	Turn on/off the debugging output to the NUT log.

Figure 105: Actions provided by UPSmon.py.

11. Texts to be included in messages may be given names, and may incorporate other named messages. The upsmon `NOTIFYMSG %` substitution is extended to provide the substitutions

shown in table 106.

%(u)s	Fully qualified name of the UPS unit
%(c)s	Current charge of the UPS unit
%(e)s	The event which has produced this message
%(b)s	A banner of the form "2020-08-15 upsd@bigbox"
%(h)s	The hostname, the name of the local machine

Figure 106: % substitutions available in messages.

12. The low battery status LB provided by upsd is supplemented by three further low battery statuses LB1, LB2 and LB3 for which the trip levels may be set in UPSmon.conf.

12.2 Compatibility with upsmon.

UPSmon.py can be run at the same time and in the same machine as upsmon. UPSmon.py does not interfere with direct access to upsd port 3493. Command line utility programs such as upsc still function notmally.

12.3 Overview of UPSmon.py

The script has a configuration file, and many options. In general few options and in some simple cases none at all need be changed. To see the options and their default values you can enter command UPSmon.py --help

```
839 $ UPSmon.py --help
840 usage: UPSmon.py  [-h] [-c <file>] [-l <file>]
841   [-n <executable>] [-w <executable>] [-u <user>]
842   [--upsdtimeout <float>] [--command fsd|reload|stop]
843   [--sudo <executable>] [--shell <shell>]
844   [-D] [-Y] [-K] [-v]
```

Figure 107: Command UPSmon.py --help

.

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

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

-D, --debug Increase the debugging level, may be repeated but then you get more than any human can read. Debugging output is written into a NUT log file. This option does not cover Lex and Yacc.

`-Y, --debugYacc` Increase the debugging level for Lex and Yacc. No human being should ever be required to read this stuff. Debugging output is written into a NUT log file.

`-c <file>, --config <file>` The configuration file. UPSmon.py tries to guess where you put this. Debian sysadmins might see `/etc/nut/UPSmon.conf` . OpenSUSE admins might see `/etc/ups/...` See table 126 for a list of possible directories.

`-l <file>, --logfile <file>` The log file, with default `/var/log/NUT.log` Progress and error messages and the stuff generated by options `-D` and `-Y` go into this file. Note that if upsdTLS.py and UPSmon.py are running in the same machine they will write into the same log. See chapter 26.3 for an extension to `logrotate` to cover this file.

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

`-n <executable>, --notify <executable>` The notification executable. The default is `/usr/bin/notify-send -t 0 -u critical`

`-w <executable>, --wall <executable>` The wall executable. The default is `/usr/bin/wall`

`-u <user>, --user <user>` After launch as root, run as this user. UPSmon.py tries to guess the user. OpenSUSE admins would probably see `upsd`, whereas Debian admins would see `nut`. See table 126 for a list of possible users.

`--upsdtimeout <float>` Socket timeout for exchanges with upsd. The default is 1.8 seconds. Note that UPSmon.py sometimes relies on timeouts to determine that a message exchange has completed.

`--sudo <executable>` Authorise user to execute code as another user. The default is `/usr/bin/sudo` Use of `sudo` assumes that file `/etc/sudoers` allows the caller to sudo as the required user. For example

```
nut LAN = (ALL) NOPASSWD:SETENV: /usr/bin/notify-send, /usr/bin/wall
nut LAN = (ALL) NOPASSWD:SETENV: /sbin/shutdown
```

where LAN is defined by a declaration such as

```
Host_Alias LAN = 10.218.0/255.255.255.0, 127.0.0.1, localhost
```

To update `/etc/sudoers` use `visudo` , for example `VISUAL=/usr/bin/emacs visudo -f /etc/sudoers`

`--shell <file>` The shell that will process the `SHELLCMD` actions. The default is `/bin/bash -c`

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

12.4 Running UPSmon.py

It is possible, in a simple installation, to run the daemon UPSmon.py in the same machine as upsd. However the design is for remote monitoring of one or more upsd servers across an un unreliable network. UPSmon.py assumes that the server(s) is/are already running[13] and ready to receive the `STARTTLS` message.

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

Copy the file `/usr/lib/systemd/system/nut-monitor.service` to `/etc/systemd/system/nut-py-monitor.service` and modify the new file shown in figure 108. Lines 847, 849 and 850 have been changed.

```
845 [Unit]
846 Description=Network UPS Tools - Python - power device monitor
847 After=local-fs.target network.target

848 [Service]
849 ExecStart=/usr/sbin/UPSmon.py
850 PIDFile=/var/run/UPSmon.pid
851 Type=forking

852 [Install]
853 WantedBy=multi-user.target
```

Figure 108: systemd service unit `nut-py-monitor.service` for UPSmon.py.

You may choose to place the UPSmon.py script in directory `/usr/sbin` or make `/usr/sbin/UPSmon.py` a link to wherever you put the Python script. Note that systemd service units in `/etc` take precedence over those in `/usr/lib`. See `man systemd.unit(5)`. After you have made the changes, you should run the command `systemctl daemon-reload` . See `man systemctl(1)`. Before running upsdTLS.py the first time, you will need to run the command

```
systemctl enable nut-py-monitor.service
```

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

`systemctl daemon-reload` to make any changes to the service unit available to systemd.

`systemctl enable nut-py-monitor.service` to make the daemon UPSmon.py operational and "startable".

[13] The general case is for further work.

`systemctl start nut-py-monitor.service` to start UPSmon.py. Note that this will not erase the log file. If you want to clear the log file then you need to do that yourself. See also chapter 26.3 for a discussion of log rotation.

`systemctl status nut-py-monitor.service` to see the current status of daemon UPSmon.py.

`systemctl stop nut-py-monitor.service` to stop UPSmon.py.

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

Serious errors will prevent UPSmon.py from starting and you can read about them in the NUT log and in the system log. After starting UPSmon.py, check the NUT log for warnings and other error messages. Look for the reports beginning "`Sanity checks for this configuration ...`".

12.5 UPSmon.py's status changes

EVENTS based on upsd status changes		
None->ALARM	ALARM->None	The UPS has raised/dropped the alarm signal.
None->BOOST	BOOST->None	The UPS is now boosting/not boosting the output voltage.
None->BYPASS	BYPASS->None	The UPS is/is not now bypassing its own batteries.
None->CAL	CAL->None	The UPS is/is not now in calibration mode.
None->CHRG	CHRG->None	The UPS is/is not now recharging its batteries.
None->DISCHRG	DISCHRG->None	The UPS is/is not now discharging its batteries.
None->LB	LB->None	The driver says the UPS battery charge is now low/no longer low with respect to level defined by upsrw. See chapter 2.10.
None->OFF	OFF->None	The driver says the UPS is/is not now OFF.
OL->OB	OB->OL	The UPS is now on battery/no longer on battery.
None->OVER	OVER->None	The UPS is/is not now in status [OVER].
None->RB	RB->None	The UPS needs/no longer needs to have its battery replaced.
None->TEST	TEST->None	The UPS is/is not now performing a test.
None->TRIM	TRIM->None	The UPS is now trimming/not trimming the output voltage.
Other EVENTS monitored by UPSmon.py		
COMM->NOCOMM	NOCOMM->COMM	Communication with the UPS in now lost/restored.
None->LB1	LB1->None	The UPS battery charge is now low/no longer low with respect to level L defined by declaration `LET battery.charge.low.1 = 'L'`.
None->LB2	LB2->None	The UPS battery charge is now low/no longer low with respect to level L defined by declaration `LET battery.charge.low.2 = 'L'`.
None->LB3	LB3->None	The UPS battery charge is now low/no longer low with respect to level L defined by declaration `LET battery.charge.low.3 = 'L'`.
None->FSD	FSD->None	The UPS is/is not now in Forced ShutDown mode.
None->TICK	TICK->None	A heartbeat UPS has/has not generated a [TICK].
None->TOCK	TOCK->None	A heartbeat UPS has/has not generated a [TOCK].
TO->my-timer		Timer "my-timer" has completed.

Figure 109: Symbols used to represent events monitored by UPSmon.py.

UPSmon.py, like NUT's upsmon is an example of a client of upsd[14]. Just as upsmon does, it runs permanently as a daemon in a local or remote box, polling the status changes of the UPS unit. It is able to react to changes in the UPS state for example by emitting warning messages, or shutting down the box. The actions are specified in the configuration file `UPSmon.conf` which will be discussed in specific examples.

As the state of a UPS evolves, each status change, called an "EVENT", is identified with the symbols shown in figure 109. *(These correspond to the NOTIFY events, also known as a "notifytype" in NUT.)*

Figure 104 shows what happens when wall power fails. Daemon upsd has polled the UPS, and has discovered that the UPS is supplying power from it's battery. The `ups.status` changes to [OB]. Daemon UPSmon.py has polled upsd, has discovered the status change and has generated the OL ->OB event.

12.6 Configuration file

There is just one configuration file for UPSmon.py which replaces `upsmon.conf`, `upssched.conf` and upssched-cmd. The formal grammar for this configuration file is in chapter 26. The file contains:

1. Comments and blank lines. A comment begins with a # character found outside a quoted text, and continues up the the end-of-line.
2. Initial declarations. See section 12.6.1
3. One or more group declarations. See section 12.6.2

The following technical terms are used in the descriptions of the configuration file:

quotation mark One of the following five styles[15] of text marker. See chapter 25 for help in typing the characters which may not be on your keyboard.

1. double quotation marks: "*bla..bla...*" which are probably on your keyboard,
2. single quotation marks: '*bla..bla...*' which are also on your keyboard,
3. french guillemets: «*bla..bla...*»,
4. mathematical left ceiling/right floor ⌈*bla..bla...*⌋ and
5. corner brackets used for quotations in asian lanuages: ⌜*bla...bla...*⌟.

quotetext A text in quotation marks. E.g. «Hello World»

quotetexts A sequence of one or more *quotetext* declarations. E.g. «Today is » «Friday.» This results in a single text "Today is Friday."

[14] See chapter 1.3.2 for details of upsd.

[15] I couldn't decide which ones to use so I kept them all. Ed.

number An integer or floating point number such as 15 or 2.8.

name Names for groups, timers, UPS's, messages. The name begins with `[a-zA-Z_]` and continues with as many of `[a-zA-Z0-9._%+-:@]` as you like. E.g. `UPS31.a@BIG_BOX`.

ups-name All UPS's must be individually identified. Unlike NUT, there are no "wildcard" UPS's. Each UPS has a formal "fully qualified" name which is of the form *group*`:`*ups*`@`*host*`:`*port* , for example `HB:heartbeat@bigbox:3493` , although shortened forms are used where there is no ambiguity.

12.6.1 Initial declarations

The initial declarations are

`SMTPSERVER` *quotetext* `PORT` *number* `USER` *quotetext* `PASSWORD` *quotetext* If you want to send e-mails, you must provide details of your e-mail service provider. For example `SMTPSERVER 'mail.gandi.net' PORT 465 USER 'mbox@example.com' PASSWORD «1234»` . Connections with the SMTP server are always TLS encrypted.

`LET` *name* `=` *quotetexts* Provide a name for one or more *quotetext*. This saves a lot a typing. For example `LET banner = ⌜[%(b)s] UPS=%(u)s charge=%(c)s event=%(e)s⌟` . The named message `LET hostname =` *hostname* is built in.

`MAXNOTIFY` *number* This limits the number of on-screen notifications, and was needed during early debugging when things often exploded. It will probably be removed in the future. The default is 20.

`POLLFREQ` *number* This is the polling period for all UPS units managed by this UPSmon.py instance. The default, which is the recommended value is 5 seconds. See also `man upsmon.conf`

`POLLFREQALERT` *number* This is the polling period for all UPS units managed by this UPSmon.py instance when any one of them is in status [OB]. The default is 5 seconds.

12.6.2 Group declarations

The group declarations are a sequence of one or more `GROUP` which are structured as follows:

`GROUP` *name* `HOST` *name* `PORT` *number* `CERTFILE` *name/quotetext* One or more UPS units share the same `HOST`, `PORT` and TLS `CERTFILE`. E.g. `GROUP LOCAL HOST localhost PORT 401 CERTFILE monitor.cert.pem` . The UPS units attached to this host are grouped together and each is specified by a `MONITOR` declaration in this group.

`LET` *`name = quotetexts`* Further named texts. Note that there is only one name space shared by all `LET` declarations. It's up to you to avoid clashes. The *name* `battery.charge.low.`*i* for $i = 1..3$ is a special case in which the *quotetexts* must be quoted integer. The effect is to assign the integer value as the battery charge level at which the events None->LB*i* and LB*i*->None will occur. For example `LET battery.charge.low.2 = '33'` The level is set for the most recently defined UPS, i.e. the previous `MONITOR` declaration. The default levels are LB1=50, LB2=25 and LB3=12.

`MONITOR` *`ups-name`* `POWERVAL` *`number`* `UPSDUSER` *`name`* `PASSWORD` *`quotetext`* `TYPE` *`name`*
Each UPS unit to be managed must be declared. The *ups-name* must match the name in the `ups.conf` declaration. See for example line 32. The `POWERVAL` is the number of power supplies that this UPS feeds. The `UPSDUSER` is the "user" declared in `upsd.users`. See line 40. The `PASSWORD` is the value declared in `upsd.users`. See line 41. The `TYPE` value must be `master` or `slave`. In NUT's `upsmon.conf` `master` means this system will shutdown last, allowing any slaves time to shutdown first. The declaration is included here to facilitate interworking with upsmon but in UPSmon.py, it is merely a declaration of intention, since the logic is decided by the declared actions.

E.g. `MONITOR ups1 POWERVAL 1 UPSDUSER leboss PASSWORD 'sekret' TYPE master`

`MINSUPPLIES` ***`number`***

Declare for each `GROUP` the number of power supplies which must be operational, and that if fewer are available, NUT must shut down the server. The default value is 1 if this declaration is omitted. See chapter 3.2

More work needed here to create a MINSUPPLIES event.

12.6.3 Action declarations

`WHEN` *`ups-name`* `REPORTS` *`old-status -> new-status : actions`*
Declare what, if anything, is to be done when an event, i.e. a status change occurs. The *ups-name* may be abbreviated when there is no ambiguity, but the fully qualified UPS name is always used internally.

Both *old-status* and *new-status* are one of `ALARM`, `BOOST`, `BYPASS`, `CAL`, `CHRG`, `COMM`, `DISCHRG`, `FSD`, `LB`, `NOCOMM`, `OFF`, `OB`, `OL`, `OVER`, `RB`, `TEST`, `TICK`, `TOCK`, `TRIM` and `None`.

The sequence *old-status* `->` *new-status* defines a status change, i.e. an event. The valid events are listed in chapter 12.5.

When the event specified for this UPS is detected, the *actions* will be executed. For example `WHEN ups1 REPORTS None->LB :` *`actions`* Let's hope those actions do something useful.

`WHEN` *`ups-name`* `TIMEOUT` *`timer-name : actions`*
Declare what, if anything, is to be done when a timeout occurs. The *timer-name* will have

been declared by a previous `STARTTIMER` action. `TIMEOUT` may be written as `TO`. For example `WHEN ups1 TO final-delay : SHUTDOWNCMD «/sbin/shutdown -h now»`

actions A sequence of one or more of the following:

condition `CANCELTIMER` ***timer-name*** The *timer-name* must have been declared by a previous `STARTTIMER` action. It is not an error to cancel a timer after it has run out.

condition `DEBUG 0/1/2` Initiate or terminate debugging output. Note that since a set of actions is executed in random order, you should not rely on a `DEBUG` in the same set of actions as the action you wish to trace.

condition `EMAIL FROM` ***quotetext*** `TO` ***quotetext*** `SUBJECT` ***quotetext*** `MESSAGE` ***quotetexts*** Send an email via the mail server declared in the introduction by `SMTPSERVER`. E.g.

```
EMAIL FROM «UPSmon.py@example.com»
      TO   «sysadmin@bigbox.com»
      SUBJECT «Msg-1-min»
      MESSAGE «Msg-1-min»
```

Where `Msg-1-min` has been previously declared in a `LET`. Note that the message must be in 7-bit ascii. Any character more exotic will be converted to a "~".

condition `STARTTIMER` ***timer-name number*** Declare and start a timer with the given name, and the given value in seconds. It is up to you to avoid name conflicts between timers and with other names. E.g. `STARTTIMER final-delay 5`

condition `EPRINT` ***quotetexts*** Send the *quotetexts* to STDERR. When UPSmon is daemonized, `EPRINT` is ignored. Use `NUTLOG` instead.

condition `NOTIFY` ***quotetexts*** Place the *quotetexts* in an on-screen notification for all logged-in users. If UPSmon.py is run as a non-privileged user, which is usually the case, than that user, for example `nut`, must be given access to program `notify-send` in file `/etc/sudoers` . See chapter 23.2 for details of how to do this. See also `man sudo(8)` for lots and lots of brain-damaging detail.

condition `NUTLOG` ***quotetexts*** Write the *quotetexts* into the NUT log file specified by option `--logfile`. The *quotetexts* will be prepended with a timestamp and a reminder of the source program and line number. For example action `NUTLOG «Hello World»` might add the following line to the log file:

```
18:32:25.164 UPSmon.py[3498] Hello World
```

See chapter 26.3 for an extension to `logrotate` to cover this file.

condition `PRINT` ***quotetexts*** Send the *quotetexts* to STDOUT. When UPSmon is daemonized, `PRINT` is ignored. Use `NUTLOG` instead.

condition `SETFSD` ***ups-name*** This action sets the "forced shutdown" flag on each slave UPS when the master plans to power it off. This is done so that slave systems will know about the power loss and shut down before the UPS power disappears. UPSmon.py, like upsmon, in master mode is the primary user of this function.

Setting this flag makes [FSD] appear for this UPS. This [FSD] should be treated just like a [OB LB]. To use this action, you need `upsmon master` in `upsd.users`, or "FSD" action granted in `upsd.users`. See `man upsd.users`.

Note that [FSD] in upsd is currently a latch - once set, there is no way to clear it short of restarting upsd. This may cause issues when upsd is running on a system that is not shut down due to the UPS event.

See the Network UPS Tools Developer Guide, Network protocol information

condition `SHELLCMD` ***quotetexts*** Call on the shell defined by the option `--shell` to execute the command given by the *quotetexts*. For example

```
SHELLCMD «echo "Today is $(date)" >> /var/log/NUT.log»
```

might write "`Today is Tue Oct 13 10:09:02 CEST 2020`" into the log file.

condition `SHUTDOWNCMD` ***quotetexts*** Call for a system shutdown using the command specified by the *quotetexts*. For example, `SHUTDOWNCMD «/sbin/shutdown -h 0»`. If UPSmon.py is run as a non-privileged user, which is usually the case, than that user, for example `nut`, must be given access to program `shutdown` in file `/etc/sudoers` . See chapter 23.2 for details of how to do this. See also `man sudo(8)` for lots of detail.

condition `SYSLOG` ***quotetexts*** Write the *quotetexts* into the system log. The system log provides 8 levels of urgency. They are shown, in order of decreasing importance, in table 110. If your *quotetexts* are prefixed with one of these urgency indicators, your message

`[emerg]`	System is unusable
`[alert]`	Action must be taken immediately
`[crit]`	Critical conditions
`[err]`	Error conditions
`[warning]`	Warning conditions
`[notice]`	Normal, but significant, condition
`[info]`	Informational message (default)
`[debug]`	Debug-level message

Figure 110: System log urgency levels.

will be logged at the required level e.g. `SYSLOG «[debug]» « UPS %(u)s burning»` . The default level is `[info]`.

condition* `WALL` *quotetexts Place the *quotetexts* in a console message for all logged-in users. If UPSmon.py is run as a non-privileged user, which is usually the case, than that user, for example `nut`, must be given access to program `wall` in file `/etc/sudoers` . See chapter 23.2 for details of how to do this. See also `man sudo(8)` for details. Note that `wall` does not support UTF-8.

condition This is either empty or has the form `IF` *`old-status -> new-status`*. The condition has the value True if in the sequence of events from the given UPS, that UPS now has status *new-status*. For example the expression `IF OB -> OL` is True if the UPS currently has status [OL] and False if the UPS has status [OB]. Note that *old-status* -> *new-status* must be a valid event as listed in chapter 12.5.

13 UPSmon.py configuration

A configuration file `UPSmon.conf` must be created to tell UPSmon.py how to handle the status changes coming from upsd. As with `upsmon.conf`, this can be done manually, but for simple cases, probably the majority, in which upsd and UPSmon.py run in the same machine, UPSmon.py provides a Python3 tool mkUPSmonconf.py, to create a complete fully functioning configuration file. You can either use the output of this tool or take it as the starting point for a customised configuration.

13.1 Configuration tool mkUPSmonconf.py

mkUPSmonconf.py is a Python3 script which will build a simple configuration file `UPSmon.conf` for UPSmon.py. The output is to STDOUT. The status is "experimental". The script is intended for demonstration and experiment. The license is GPL v3 or later at your choice, with support in the "ups-user" mailing list. There is documentation

The script has options which you select to introduce site-specific data. You have to specify all the options. To see the options to be specified you can enter command `mkUPSmonconf.py --help`

```
$ mkUPSmonconf.py --help
usage: mkUPSmonconf.py [-h] [--plan standard|timed]
  [--ups <name>] [--upsdname <name>] [--upsdport <integer>]
  [--certfile <filename>] [--upsduser <name>]
  [--upsdpass <string>] [--smtpserver <domain>]
  [--smtpport <integer>] [--smtpuser <name>]
  [--smtppass <string>] [--emailfrom <string>]
  [--emailto <string>] [-v]
```

Figure 111: Command `mkUPSmonconf.py --help`

.

Let's look at these arguments in more detail.

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

`--plan standard|timed` Specify standard or timed shutdown plan. Valid options are `standard` or `timed`.

`--ups` <***name***> The name of your UPS, for example `UPS_123`. If you have more than one UPS unit then create a configuration file for the first, and then extend it using copy/paste of the actions for the second.

`--upsdname` <***name***> The name of the system on which upsd runs. E.g. `localhost` if UPSmon.py and upsd run on the same machine.

`--upsdport` <***integer***> The TLS port used by upsd/upsdTLS.py. E.g. `401`

`--certfile` <***filename***> The file which holds the public TLS certificate for upsd/upsdTLS.py. E.g. `/etc/nut/bigbox-monitor.cert.pem`

`--upsduser` <***name***> User for this UPS, as given in `upsd.users`. E.g. `upsmaster` on line 40

`--upsdpass` <***string***> The password for this upsd user, as given in `upsd.users`. E.g. `password = sekret` on line 41

`--smtpserver` <***domain***> Your e-mail server. E.g. `mailbox.mailserver.com`

`--smtpport` <***integer***> Your e-mail server's TLS port. E.g. `465` . Communication with the mail server is always TLS encrypted.

`--smtpuser` <***name***> Your sign-in account name on the e-mail server. E.g. `mailbox@mydomain.com`

`--smtppass` <***string***> The password for your account on the e-mail server. E.g. `qwertyuiop`

`--emailfrom` <***string***> The email address from which messages will be sent. E.g. `"<bigserver@bigU.edu>"` Note the email convention of placing the address in angle brackets, and the double quotes needed to prevent Bash from interpreting the angle brackets.

`--emailto` <***string***> The email address of the person to whom messages will be sent. E.g. `"Big Joe <jschmoe@bigU.edu>"` Note the email convention of placing the address in angle brackets, and the double quotes needed to prevent Bash from interpreting the angle brackets.

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

13.2 Using configuration tool mkUPSmonconf.py

Call the program from the command line. If you forget an option you will get a message such as "You have forgotten to specify option --smtppass". A typical call is

```
862 mkUPSmonconf.py\
863     --plan timed --ups Eaton --upsdname localhost --upsdport 401\
864     --certfile /etc/ups/mkNUTcert/titan-monitor.cert.pem\
865     --upsduser upsmaster --upsdpass sekret --smtpserver mail.gandi.net\
866     --smtpport 465 --smtpuser mailbox@rogerprice.org\
867     --smtppass qwertyuiop --emailfrom "<UPSmon@rogerprice.org>"\
868     --emailto "Roger Price <roger@rogerprice.org>" > /etc/nut/UPSmon.conf
```

Figure 112: Calling mkUPSmonconf.py

If you will be typing this several times, you might want to put the command in a shell script. Note on line 868 that the output is directed to file `/etc/nut/UPSmon.conf`. Note also on lines 867 and 868 that the values for options `--emailfrom` and `--emailto` have to be quoted to prevent Bash from interpreting what it would consider to be < and > redirections.

13.3 UPSmon.conf configuration examples

Let's look at a shutdown plan generated by mkUPSmonconf.py.

13.3.1 Timed shutdown plan, part 1 of 4, the introduction

```
869 # UPSmon.conf timed shutdown plan generated by mkUPSmonconf.py version 1.0
                on 2020-10-14T14:36:42.344212
870 # Python version 3.4.6 (default, Mar 22 2017, 12:26:13) [GCC] running on titan
871 # Calling command:
     ./mkUPSmonconf.py --plan timed --ups Eaton --upsdname localhost
         --upsdport 401 --certfile /etc/ups/mkNUTcert/titan-monitor.cert.pem
         --upsduser upsmaster --upsdpass sekret --smtpserver mail.gandi.net
         --smtpport 465 --smtpuser mailbox@rogerprice.org --smtppass qwertyuiop
         --emailfrom <UPSmon@rogerprice.org> --emailto Price <roger@rogerprice.org>
872 # Support: nut-upsuser mailing list.
873 # Documentation: http://rogerprice.org/NUT/ConfigExamples.A5.pdf

874 # All groups share the same POLLFREQ and POLLFREQALERT and e-mail relay
875 POLLFREQ 5.0  POLLFREQALERT 5.0
876 SMTPSERVER «mail.gandi.net» PORT 465
877      USER «mailbox@rogerprice.org» PASSWORD «qwertyuiop»

878 # Named messages  Let hostname = hostname is built in.
879 LET banner        = ⌜[%(b)s] UPS=%(u)s charge=%(c)s event=%(e)s␣
880 LET Msg-COMM      = banner "
881                      " I have re-established communication with this UPS."
882 LET Msg-NOCOMM    = banner " "I have lost communication with this UPS."
883 LET Msg-OL        = banner " Power restored, shutdown cancelled."
884 LET Msg-RB        = banner " Battery needs replacement."
885 LET Msg-shutdown  = banner " On battery, shutting down now ..."
886 LET Certfile      = «/etc/ups/mkNUTcert/titan-monitor.cert.pem»
```

Figure 113: Timed shutdown plan, part 1 of 4, the introduction.

Notes on figure 113

1. The command used to generate the file is repeated on line 871 but the quoting needed by Bash does not appear since the Python3 program does not see the quotes. If you repeat the command, you will have to re-introduce the quoting.
2. The `POLLFREQ` and `POLLFREQALERT` on line 875 are the same as upsmon. See chapter 4.1.
3. On line 876 the `PORT` number corresponds to a TLS port. Communication with the email service provider is always TLS encrypted.
4. On lines 876-877 the «...» is added automatically by the mkUPSmonconf.py script. You do not have to do this.
5. Line 886 corresponds to an OpenSUSE installation. A Debian sysadmin would probably prefer address `/etc/nut/...` See table 126 for a list of possible directories.

13.3.2 Timed shutdown plan, part 2 of 4, the shutdown

```
887 # The local UPS units
888 GROUP LOCAL HOST localhost PORT 401 CERTFILE Certfile
889 MONITOR Eaton POWERVAL 1 UPSDUSER upsmaster PASSWORD «sekret» TYPE master

890 # Timed plan specific actions
891 LET Msg-2-min = banner " On battery, shutdown in 2 mins, save your work ..."
892 LET Msg-1-min = banner " On battery, shutdown in 1 min, save your work ..."
893 WHEN Eaton REPORTS OL->OB :   NOTIFY Msg-2-min NUTLOG Msg-2-min
894                               STARTTIMER two-min 120 STARTTIMER one-min 60
895 WHEN Eaton TIMEOUT one-min :  NOTIFY Msg-1-min NUTLOG Msg-1-min WALL Msg-1-min
896                               EMAIL FROM « <UPSmon@rogerprice.org> »
897                                     TO   « Roger Price <roger@rogerprice.org> »
898                                     SUBJECT «Msg-1-min»
899                                     MESSAGE «Msg-1-min»
900 WHEN Eaton TIMEOUT two-min :  NOTIFY Msg-shutdown NUTLOG Msg-shutdown
901                               WALL Msg-shutdown STARTTIMER final-delay 5
902 WHEN Eaton REPORTS OB->OL :   NOTIFY Msg-OL NUTLOG Msg-OL WALL Msg-OL
903                               CANCELTIMER two-min CANCELTIMER one-min
                                  CANCELTIMER final-delay
904 # End of timed plan specific actions

905 # Shutdown on low battery
906 WHEN Eaton REPORTS None->LB : NOTIFY Msg-shutdown NUTLOG Msg-shutdown
907                               WALL MSG-shutdown STARTTIMER final-delay 5
908 WHEN Eaton TIMEOUT final-delay : SHUTDOWNCMD "/sbin/shutdown -h 0"
```

Figure 114: Timed shutdown plan, part 2 of 4, the shutdown.

Notes on figure 114

1. Line 888 introduces the notion of "GROUP". In general a group is a set of UPS units which are attached to the same upsd server. In NUT's upsmon.conf the MONITOR *system* declaration identifies the upsd host system and the port. See man upsmon.conf. UPSmon.conf transfers the host system and port identification to a named group, and adds the CERTFILE declaration.
2. Line 889 resembles the upsmon.conf declaration, but with the inclusion of additional keywords for clarification. "Eaton" declares the UPS name, the HOST and PORT have already been declared. The UPS name should correspond to the name specified in ups.conf. See line 32.
3. Since this is the timed plan rather than the standard plan, we need additional messages which are declared on lines 891-892.
4. When event OL->OB arrives, lines 893-894 call for the "on battery" message to be put on-screen and in the NUT log file. The actions also declare the timers two-min and one-min and start them.
5. When timer one-min runs out, lines 895-899 place warnings on screen, in the NUT log file and on all logged in terminals. The actions also send an email to the administrator.
6. When timer two-min runs out, lines 900-901 place warnings on-screen, in terminals and in the NUT log file. A short final-delay timer is declared and started. This timer corresponds to FINALDELAY in upsmon.conf.
7. What happens if power returns before the shutdown? If event OB->OL arrives, lines 902-903 notify the user, place a message in the NUT log file and turn off all the timers.
8. Whether the plan is "standard" or "timed" the local system must be shutdown on event None ->LB. This happens on lines 906-907. Users receive a final on-screen warning, a message goes into the NUT log file and the action declares and starts a short final-delay timer.
9. When the final-delay timer runs out, line 908 calls for a system shutdown.

13.3.3 Timed shutdown plan, part 3 of 4, warnings

Notes on figure 115

1. Some UPS units are capable of reporting that the battery needs replacement. On line 910, when event None->RB arrives messages are placed on-screen and in the NUT log file. Line 912 sends an email to the sysadmin. The upsmon RBWARNTIME behaviour is reproduced by defining and starting an rbwarntime timer.
2. Line 916 specifies that when the rbwarntime timer runs out, an on-screen message appears[16] and also goes into the NUT log file. The action also restarts the timer. It will continue to loop until the status [RB] disappears with event RB->None on line 917

[16] Do the users have to be told about this?

```
909 # Warning for battery replacement
910 WHEN Eaton REPORTS None->RB : STARTTIMER rbwarntime 43200
911                               NUTLOG Msg-RB NOTIFY Msg-RB
912                               EMAIL FROM « <UPSmon@rogerprice.org> »
913                                     TO   « Roger Price <roger@rogerprice.org> »
914                                     SUBJECT «Msg-RB»
915                                     MESSAGE «Msg-RB»
916 WHEN Eaton TIMEOUT rbwarntime : STARTTIMER rbwarntime 43200
                                    NUTLOG Msg-RB NOTIFY Msg-RB
917 WHEN Eaton REPORTS RB->None : CANCELTIMER rbwarntime

918 # Warning that UPSmon has lost UPS Eaton.  Shut down on NOCOMM when OB.
919 WHEN Eaton REPORTS COMM->NOCOMM : STARTTIMER nocommwarntime 300
920                                   IF OL->OB NOTIFY Msg-shutdown
921                                   IF OL->OB NUTLOG Msg-shutdown
922                                   IF OL->OB WALL Msg-shutdown
923                                   IF OL->OB STARTTIMER final-delay 5
924 WHEN Eaton TIMEOUT nocommwarntime : NUTLOG Msg-NOCOMM NOTIFY Msg-NOCOMM
925 WHEN Eaton REPORTS NOCOMM->COMM : CANCELTIMER nocommwarntime
                                   NUTLOG Msg-COMM NOTIFY Msg-COMM
```

Figure 115: Timed shutdown plan, part 3 of 4, warnings,

3. The statuses [COMM] and [NOCOMM] are not due to upsd. They are generated internally by UPSmon.py when it has problems talking to upsd. The `standard` and `timed` configurations discussed here assume that upsd and UPSmon.py are running in the same machine, but in general this is not the case, and network problems become more apparent when upsd and UPSmon.py are separated.

   The event COMM->NOCOMM starts a timer which will later place a warning message in front of users and in the NUT log file. This follows the upsmon logic. Additionally, and again following upsmon logic, a shutdown procedure will begin if the system is currently running on battery. See lines 920-923. Note that the condition must be attached to each of the actions.

   Note the subtle difference between upsmon and UPSmon.py. See figure 14. On line 68 daemon upsmon will trigger a [NOCOMM] NOTIFY event after `NOCOMMWARNTIME` seconds if it can't reach **any** of the UPS entries in configuration file `upsmon.conf`. UPSmon.py does this for each UPS individually. The difference is slight if there is only one UPS :-)

4. On line 925 the timer `nocommwarntime` is cancelled and suitable messages send to the users[17] and the NUT log file.

[17] Is it really necessary to notify the users of this technical matter?

13.3.4 Timed shutdown plan, part 4 of 4, heartbeat

```
926 # heartbeat.conf
927 # 20 minute heartbeat
928 ups.status: TICK
929 TIMER 600
930 ups.status: TOCK
931 TIMER 600
```

Figure 116: Configuration file heartbeat.conf

The NUT software runs in the background for weeks, months without difficulty and with no messages going the system administrator. "All is well!", but is it?

NUT is a collection of pieces and interconnecting protocols. What if one of these pieces has stopped or the protocol blocked? We need something that will check regularly that all is indeed well. The proposed heartbeat does this job.

Heartbeat definitions are not provided by NUT, you have to create them for yourself. Create the new file heartbeat.conf as shown in figure 116 in the same directory as ups.conf. For security, only users upsd/nut and root should have write access to this file.

The heartbeat will cycle continuously through this script.

Lines 928 and 930 flip the ups.status value between [TICK] and [TOCK].

Lines 929 and 931 place a 10 minute time interval between each status change. $2 \times 600sec = 20min$, the heartbeat period.

```
932 [heartbeat]
933   driver = dummy-ups
934   port = heartbeat.conf
935   desc = "Watch over NUT"
```

Figure 117: Addition to the file ups.conf for heartbeat.conf

You must also declare to upsd that it is to generate the heartbeat. Add the declaration shown in figure 117 to file ups.conf. In line 933 we see the driver used to generate the heartbeat. This driver is also used for debugging. You can amuse yourself by adding further status changes and observing their effect.

Notes on figure 118:

1. On line 939 a group "HB" is declared to contain the heartbeat UPS. The HOST, PORT and CERTFILE are the same as for the physical UPS.
2. Lines 940-941 declare messages specific to the heartbeat.
3. Other than the POWERVAL of 0, the MONITOR declaration on line 942 is the same as for the physical UPS.
4. Line 943 says that the heartbeat does not require electrical energy. This zero declaration also circumvents certain sanity checks that real UPS's must pass.
5. Lines 944 and 947 manage the timers which watch over the [TICK] and [TOCK] comming from upsd. The timer is longer than the expected interval between status arrivals. If this timer expires we assume that the heartbeat has failed.
6. Logging the None->TICK on linbe 946 produces a log message every 20 minutes.
7. Line 949 is a form of "goto" so all the heartbeart error logging is in one place.

8. Lines 950-954 send heartbeat failure messages to the system administrator and to the NUT log file.

```
936 # Heartbeat operation, requires file heartbeat.conf in the upsd server,
937 # and definition of UPS [heartbeat] in ups.conf.  Note that the timer
938 # specified here must be longer than the timer in heartbeat.conf.
939 GROUP HB HOST localhost PORT 401 CERTFILE Certfile
940 LET Msg-HB-start = banner " Event %(e)s Start HB-timer"
941 LET MSG-HB-fails = banner " %(u)s FAILURE."
                       "I have not received expected TIC/TOC status change."
942 MONITOR heartbeat POWERVAL 0 UPSDUSER upsmaster PASSWORD «sekret» TYPE master
943 MINSUPPLIES 0
944 WHEN heartbeat REPORTS None->TICK : CANCELTIMER tock-timer
945                                     STARTTIMER tick-timer 660
946                                     NUTLOG Msg-HB-start
947 WHEN heartbeat REPORTS None->TOCK : CANCELTIMER tick-timer
                                     STARTTIMER tock-timer 660

948 # What to do if the heartbeat fails
949 WHEN heartbeat TIMEOUT tick-timer : STARTTIMER tock-timer 0.5
950 WHEN heartbeat TIMEOUT tock-timer : NUTLOG MSG-HB-fails  NOTIFY MSG-HB-fails
951                                     EMAIL FROM « <UPSmon@rogerprice.org> »
952                                           TO   « Price <roger@rogerprice.org> »
953                                           SUBJECT «Msg-HB-fails»
954                                           MESSAGE «Msg-HB-fails»
955 # End of file
```

Figure 118: Timed shutdown plan, part 4 of 4, heartbeat.

13.3.5 Standard shutdown plan

The only differences between the standard plan and the timed shutdown plan are that the standard plan removes lines 890-904 and replaces then with lines 957-958. These actions send a warning message to the users and to the NUT log file.

```
956 # Standard plan specific actions
957 LET Msg-OB = banner " Power failure, possible shutdown, save your work ..."
958 WHEN [UPS] REPORTS OL->OB :   NOTIFY Msg-OB NUTLOG Msg-OB WALL Msg-OB
959 # End of standard plan specific actions
```

Figure 119: Standard shutdown plan differences

14 UPSmon.py installation checklist

Here is the editor's checklist of the things to do to install and run UPSmon.py.

1. Check that you have Python 3.6 running. No? You will need to install it.
2. Check that you have OpenSSL 1.1.1d or better.
3. Download UPSmon.py, upsdTLS.py, mkNUTcert.py and mkUPSmonconf.py from rogerprice.org/NUT to wherever you put Python3 scripts.
4. Review the shebangs at the top of the Python3 scripts. Modify if needed to meet the local situation. The shebangs that come with the scripts are those used by the editor. Yours may well be different.
5. Create symlink from `/sbin/UPSmon.py` to wherever you put the Python3 scripts. Create similar links for upsdTLS.py, mkNUTcert.py and mkUPSmonconf.py.
6. Install systemd service units `/etc/systemd/system/nut-py-server.service` and `/etc/systemd/system/nut-py-server.service`
7. Run `systemctl daemon-reload` and then enable the `nut-py-server` and `nut-py-monitor` service units.
8. Add programs `shutdown`, `wall` and `notify-send` to `/etc/sudoers` for users `nut`/`upsd`.
9. Run mkNUTcert.py to make TLS certificates
10. Run mkUPSmonconf.py to create the UPSmon.py configuration file.
11. Install `/etc/logrotate.d/NUT` .
12. Check that `heartbeat.conf` is installed in the upsd machine and that `ups.conf` contains a `[heartbeat]` declaration.
13. Disable and stop the `nut-monitor` service unit.
14. Enable and start the `nut-py-server` and then the `nut-py-monitor` service units.
15. Check output of command `ps -elf | grep -E "nut|upsd"` which on an openSUSE machine gives the output shown in figure 120.

```
960 1 S upsd 2873 1  9447 -       /usr/lib/ups/driver/usbhid-ups -a Eaton
961 1 S upsd 2878 1  5019 -       /usr/lib/ups/driver/dummy-ups -a heartbeat
962 1 S upsd 2882 1  5017 -       /usr/sbin/upsd
963 5 S upsd 2887 1 17189 core_s  /usr/local/bin/python3.8 -u /usr/sbin/upsdTLS.py
964 5 S upsd 2892 1 58813 -       /usr/local/bin/python3.8 -u /usr/sbin/UPSmon.py
```

Figure 120: upsd and UPSmon.py runtime processes

Questions? Try the "ups-user" mailing list.

Part 3

Appendices

20 Starting NUT

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

Figure 121: 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 122: 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 [18] 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[19].
- `MODE=standalone` This is the most common situation in which line 967 in figure 121 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`.

[18] See script `/sbin/upsd`.
[19] See script `/sbin/upsmon`.

21 Stopping NUT

21.1 Delayed UPS shutdown with NUT script

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

Chapter 2.5 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 77 if you want to change this.

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

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

The openSUSE distribution places the delayed shutdown script provided by NUT and shown in figure 123 in file `/usr/lib/systemd/system-shutdown/nutshutdown` . The Debian distribution places the script in file `/lib/systemd/system-shutdown/nutshutdown` .

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

Figure 123: UPS shutdown script `nutshutdown`.

On line 969 the call to upsmon with option `-K` checks the `POWERDOWNFLAG` defined by line 46. 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 971.

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

Figure 124: UPS shutdown script `nutshutdown` for 2 of 3 UPS's.

See also `man upsdrvctl`

21.2 Delayed UPS shutdown with a systemd service unit

The script provided by the NUT project in chapter 21.1 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 125.

```
# nut-delayed-ups-shutdown.service
[Unit]
  Description=Initiate delayed UPS shutdown
  Before=umount.target
  DefaultDependencies=no
[Service]
  Type=oneshot
  ExecStart=/usr/bin/logger -t nut-delayed-ups-shutdown\
                            "upsdrvctl shutting down UPS"
  ExecStart=/sbin/upsdrvctl shutdown   # Debian
[Install]
  WantedBy=final.target
```

Figure 125: UPS shutdown service unit `nut-delayed-ups-shutdown.service`.

The `ExecStart` directive on line 980 will shutdown [20] 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 980 you should specify the required UPS's as shown in lines 983-984.

```
  ExecStart=/sbin/upsdrvctl shutdown UPS-2   # Debian
  ExecStart=/sbin/upsdrvctl shutdown UPS-3
```

Note that this service unit does not perform the `upsmon -K` test for the `POWERDOWNFLAG`.

The position of this service unit may vary from one distribution to another, see section "unit file load path" in `man systemd.unit(5)`. For example in the openSUSE and Debian distributions, `/etc/systemd/system` is for a user's scripts, and `/usr/lib/systemd/system-shutdown` is for system scripts. You might use the `/etc/systemd/system` directory if your script is not part of an officially distributed product.

If you install or change this service unit, run command `systemctl --system reenable /etc/systemd/system/nut-delayed-ups-shutdown.service` . Maybe your distribution offers a graphical manager to do this.

For gory details see the systemd documentation. There are over 200 man pages starting with an index. For details of the directories used, see section "unit file load path" in `man systemd.unit`.

[20] 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.

22 Users and Directories for NUT

NUT normally runs as a non-root user, however the user varies from one distribution to another. Table 126 shows a list of users for a range of distributions.

Similarly, the configuration files used by NUT such as `upsd.conf` are placed in a directory which depends on the distribution. Table 126 also shows the directories used by different distributions.

Distribution	ID	User	Directory	ID source
Aix	aix	nut ?	/etc/nut/ ?	uname -a
Amazon	amzn	nut	/etc/ups/ ?	/etc/os-release
Arch	arch	nut	/etc/nut/	/etc/os-release
CentOS	centos	nut	/etc/ups/	/etc/os-release
Apple	darwin	nut	/etc/nut/	uname -a
Debian	debian	nut	/etc/nut/	/etc/os-release
Fedora	fedora	nut	/etc/ups/	/etc/os-release
FreeBSD	freebsd	uucp	/usr/local/etc/nut/	uname -a
Gentoo	gentoo	nut	/etc/nut/	/etc/gentoo-release
HP-UX	hpux	nut ?	/etc/nut/ ?	uname -a
IPFire	ipfire	nutmon	/etc/nut/	uname -a
Kali	kali	nut	/etc/nut/	/etc/os-release
Mint	linuxmint	nut	/etc/nut/	/etc/os-release
Apple	mac	nut ?	/etc/nut/ ?	uname -a
Mageia	mageia	nut	/etc/nut/	/etc/os-release
Manjaro	manjaro	nut	/etc/nut/	/etc/os-release
NetBSD	netbsd	nut ?	/etc/nut/ ?	uname -a
Oracle	ol	nut	/etc/ups/	/etc/os-release
OpenBSD	openbsd	ups	/etc/nut/	uname -a
OpenIndiana	openindiana	nut	/etc/nut/	uname -a
OpenSUSE	opensuse	upsd	/etc/ups/	/etc/os-release
Raspbian	raspbian	nut	/etc/nut/	/etc/os-release
Red Hat	rhel	nut	/etc/ups/	/etc/os-release
Slackware	slackware	nut	/etc/nut/	/etc/os-release
SUSE	sles	upsd	/etc/ups/	/etc/os-release
SUSE+SAP	sles_sap	upsd	/etc/ups/	/etc/os-release
Synology	synology	root ?	/usr/syno/etc/nut/	uname -a
Ubuntu	ubuntu	nut	/etc/nut/	/etc/os-release

The editor will be very pleased to hear of errors or omissions in this table.

Figure 126: Users and directories for NUT.

Notes:

1. If NUT is built without specifying the user, then the user is `nobody:nobody`.
2. FreeNAS identifies itself in `/etc/os-release` as FreeBSD.
3. The IPFire wiki suggests user `nutmon` for upsmon but makes no mention of upsd.
4. OpenIndiana: historically, NUT was not included as a package in OpenIndiana, and an OpenIndiana Wiki entry dated 2013 recommended user `ups` and directory `/opt/nut/etc/`. The values in the table are taken from OpenIndiana's current Github data for NUT.

23 Using notify-send

The program "wall" used by NUT to put notifications in front of the users is now well past it's best-before date and hardly fit for purpose. It has not been internationalized, does not support accented letters or non-latin characters, and is ignored by popular desktop environments such as Xfce, Gnome and KDE. It's apparent replacement notify-send gives the impression that it has never been tested in any other than the simplest cases, and that it is not ready for industrial strength use. Getting notify-send to work with NUT is not immediately evident, so although notify-send is not a part of NUT, we discuss this problem here.

Figure 127: Example of a notification.

23.1 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.

23.2 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". See table 126 for a list of possible users. 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`

```
985 # Host alias specification
986 Host_Alias LAN = 10.218.0/255.255.255.0,127.0.0.1,localhost,gold
987
988 upsd  LAN = (ALL)  NOPASSWD:SETENV: /usr/bin/notify-send
```

Figure 128: Modifications to file `/etc/sudoers`fig:notify.sudoer

Line 986 corresponds to the editor's system and should be adapted to your setup.
On line 988 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.

23.3 Search for and notify logged in users

Figure 129 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`.

```
989  #! /bin/bash -u
990  # notify-send-all sends notifications to all X displays
991  # Assumes /etc/sudoers allows caller to sudo as any user.
992  # E.g. nut LAN = (ALL) NOPASSWD:SETENV: /usr/bin/notify-send
993  # Call with text to be displayed as argument.
994  XUSERS=( $( who | grep -E "\(:[0-9](\.[0-9])*\)" \
995           | awk '{print $1$NF}' | sort -u ) )
996  for XUSER in $XUSERS      # E.g. jschmo(:0)
997  do NAME=(${XUSER/\(/ })   # Insert space, make NAME an array
998     DISPLAY=${NAME[1]/)/}  # E.g. :0
999     sudo -u ${NAME[0]} DISPLAY=${DISPLAY} \
1000           /usr/bin/notify-send -t 0 -u critical "$@"; RC=$?
1001    if [[ $RC -ne 0 ]]; then exit $RC; fi
1002 done
```

Figure 129: Bash script notify-send-all

Line 994 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 997 creates a Bash array containing the user name and the X display number in the form `jschmo :0)`.

Line 998 extracts the X display number `:0` and on line 999 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.

23.4 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 200.

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):

```
1003 upsd  2635     1  ... /usr/bin/usbhid-ups -a Eaton
1004 upsd  2637     1  ... /usr/bin/dummy-ups -a heartbeat
1005 upsd  2641     1  ... /usr/sbin/upsd
1006 root  2645     1  ... /usr/sbin/upsmon
1007 upsd  2646  2645  ... /usr/sbin/upsmon
1008 upsd  3217     1  ... /usr/sbin/upssched UPS Eaton@localhost: On battery
1009 upsd  3236     1  ... dbus-launch --autolaunch=d1cd...ca5d2 ...
1010 upsd  3237     1  ... /bin/dbus-daemon --fork --print-pid 5 ...
1011 upsd  3241     1  ... /usr/lib/xfce4/notifyd/xfce4-notifyd
1012 upsd  3243     1  ... /usr/lib/xfce4/xfconf/xfconfd
```

Lines 1003-1008 are due to NUT activity, and lines 1009-1012 are due to the use of notify-send. Note on line 1011 that the `xfce4-notifyd` daemon is running as user "upsd"!

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

24 Building OpenSSL and Python

The UPSmon.py program is written in Python and uses OpenSSL to make encrypted connections from the monitoring system to the system running upsd. The TLS functions of OpenSSL are updated frequently and if you want up-to-date encrypted connections, you will need recent versions of OpenSSL and Python. If you can get these using the packages of your distribution, so much the better. Otherwise you will have to build for yourself. This is not straightforward, especially for Debian.

24.1 Building OpenSSL

For the latest instructions on downloading and building OpenSSL, see "Compilation and Installation" in the Wiki. Th current version of OpenSSL installed, if any, may be seen with the command `openssl version` . For an up to date installation, the editor followed the path of least resistance: download the source, unpack it and run

```
1013 ./config
1014 make clean
1015 make
1016 make test
1017 make install
```

A careful sysadmin may well want to replace each of commands shown in lines 1013-1017 with commands such as `script -c "./config" config.log` to gather a record of what happened. If you test this as shown in line 1018

```
1018 # openssl version
1019 openssl: error while loading shared libraries:
         libssl.so.1.1: cannot open shared object file:
         No such file or directory
```

you will get the error message shown in line 1019. For Debian (stretch), you will need to add the symbolic links shown in lines 1020-1021 to reveal where you have put the OpenSSL libraries.

```
1020 ln -s /usr/local/lib/libssl.so.1.1
         /usr/lib/x86_64-linux-gnu/libssl.so.1.1
1021 ln -s /usr/local/lib/libcrypto.so.1.1
         /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
```

For openSUSE, you will need to add symbolic links shown in lines 1022-1023 to declare to the operating system where you have put the OpenSSL libraries.

```
1022 ln -s /usr/local/lib64/libssl.so
         /lib64/libssl.so.1.1
1023 ln -s /usr/local/lib64/libcrypto.so
         /lib64/libcrypto.so.1.1
```

To check that the link is correct, use the command:

```
1024 # openssl version
1025 OpenSSL 1.1.1d  10 Sep 2019
```

Well done!

24.2 Building Python

For the latest on downloading and building Python, see the Python instructions. As an example, the editor downloaded Python 3.8.1, built it and tried to install it using commands

```
1026 ./configure
1027 make clean
1028 make
1029 make altinstall
```

Line 1029 specifies `altinstall` in order to protect existing Python installations of earlier versions. A careful sysadmin may well want to replace each of commands shown in lines 1026-1029 with commands such as `script -c "./configure" configure.log` to gather a record of what happened.

Check that the `configure` program has successfully detected your new OpenSSL. You should see something like:

```
1030 checking for openssl/ssl.h in /usr/local/ssl... no
1031 checking for openssl/ssl.h in /usr/lib/ssl... no
1032 checking for openssl/ssl.h in /usr/ssl... no
1033 checking for openssl/ssl.h in /usr/pkg... no
1034 checking for openssl/ssl.h in /usr/local... yes
1035 checking whether compiling and linking against OpenSSL works... yes
1036 checking for X509_VERIFY_PARAM_set1_host in libssl... yes
1037 checking for --with-ssl-default-suites... python
```

where lines 1035-1036 are essential for a successful build. If `X509_VERIFY_PARAM_set1_host` is not found in `libssl` then `configure` needs help. This is a well known problem, see Python issue 34038. I followed the advice of *joahking* and tried the command

```
1038 script -c "./configure
       CFLAGS='-I/tmp/OpenSSL/openssl-1.1.1d/include/openssl/'
       LDFLAGS='-L/tmp/OpenSSL/openssl-1.1.1d/'"
       configure.log
```

in which `/tmp/OpenSSL` is the directory into which I downloaded OpenSSL. You will have to specify the directory you used. With this, I got the success shown in lines 1035-1036.

After `make` on Debian, you may find the following lines at the end of the `make` output:

```
1039 Could not build the ssl module!
1040 Python requires an OpenSSL 1.0.2 or 1.1 compatible
        libssl with X509_VERIFY_PARAM_set1_host().
1041 LibreSSL 2.6.4 and earlier do not provide the necessary APIs,
        https://github.com/libressl-portable/portable/issues/381
```

even though the command `openssl version` reports `OpenSSL 1.1.0l 10 Sep 2019` . You need to go back to `./configure` and check your log file.

The editor's `make install` failed with message

```
1042 zipimport.ZipImportError: can't decompress data; zlib not available
1043 Makefile:1186: recipe for target 'install' failed
1044 make: *** [install] Error 1
```

but strangely this didn't seem to affect the use of the installation for UPSmon.py.

The first attempt to run Python produces

```
1045 Could not find platform dependent libraries <exec_prefix>
1046 Consider setting $PYTHONHOME to <prefix>[:<exec_prefix>]
1047 Python 3.8.1 (default, Feb 11 2020, 22:08:59)
```

Executing command `PYTHONHOME="/usr/local" python3.8` produces

```
1048 Python 3.8.1 (default, Feb 11 2020, 22:08:59)
1049 [GCC 4.8.5] on linux
1050 Type "help", "copyright", "credits" or "license" for more information.
1051 Traceback (most recent call last):
1052   File "/etc/pythonstart", line 7, in <module>
1053     import readline
1054 ModuleNotFoundError: No module named 'readline'
```

For openSUSE, this can be fixed with a symbolic link shown at line 1055. See openSUSE 42.3 bug report 34058 https://bugs.python.org/issue34058

```
1055 ln -s /usr/local/lib64/python3.8/lib-dynload/ \
        /usr/local/lib/python3.8/lib-dynload
```

and now command `python3.8` (without setting `$PYTHONHOME`) gives

```
1056 Python 3.8.1 (default, Feb 11 2020, 22:08:59)
1057 [GCC 4.8.5] on linux
1058 Type "help", "copyright", "credits" or "license" for more information.
1059 >>>
```

There may be options for Python's `./configure` which avoid having to manually enter the symbolic link.

To check that the Python-OpenSSL setup is correct:

```
# python3.8
Python 3.8.1 (default, Feb 11 2020, 22:08:59)
[GCC 4.8.5] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import ssl
>>> ssl.OPENSSL_VERSION
'OpenSSL 1.1.1d  10 Sep 2019'
```

24.2.1 Python Lex Yacc (PLY)

You will also need to install David M. Beazly's PLY (Python Lex-Yacc).

25 Typing alternative text bracketing characters

Text in `UPSmon.conf` must be in brackets. You are free to choose which style; the following table may help you to type styles which are not on your keyboard.

Unicode		Emacs	Vim	Full name
"	U+0022	Keyboard "	Keyboard "	QUOTATION MARK (Used left and right)
’	U+0027	Keyboard ’	Keyboard ’	APOSTROPHE (Used left and right)
«	U+00AB	`AltGr{` or `Ctl-q 00ab`	`AltGr{` or `Ctl-v u00ab`	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
»	U+00BB	`AltGr}` or `Ctl-q 00bb`	`AltGr}` or `Ctl-v u00bb`	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
⎡	U+23A1	`Ctl-q 23a1`	`Ctl-v u23a1`	LEFT SQUARE BRACKET UPPER CORNER
⎦	U+23A6	`Ctl-q 23a6`	`Ctl-v u23a6`	RIGHT SQUARE BRACKET LOWER CORNER
⌜	U+2E22	`Ctl-q 2e22`	`Ctl-v u2e22`	TOP LEFT HALF BRACKET
⌟	U+2E25	`Ctl-q 2e25`	`Ctl-v u2e25`	BOTTOM RIGHT HALF BRACKET

Figure 130: Alternative text bracketing characters.

26 Grammar for UPSmon.conf

The UPSmon.conf file is parsed using David Beazley's PLY[21]. This is a pure Python approach to Lex and Yacc. There are no separate Lex and Yacc files. For background reading see "*lex & yacc*" by John R. Irvine, Tony Mason and Doug Brown, O'Reilly, first published 1990, ISBN: 1-56592-000-7.

The PLY's Lex and Yacc produce an abstract syntax tree known as AST. This is then interpreted as instructions to create a new configuration. If there are no errors, the new configuration is passed to UPSmon.py, otherwise UPSmon.py continues with the previous configuration. You can see AST in the log file if you run UPSmon.py with option -D.

26.1 Lexical structure

The configuration file is assumed to be encoded in UTF-8, and contains comments, tokens (keywords and symbols), numbers and quoted text interspersed with white space.

Whitespace Whitespace is any combination of the characters space and tab. Whitespace serves only to separate the other components of a configuration file.

Comments The character # outside a quoted text begins a comment which continues up to the end of the line. The comment is ignored by the parser. A # inside a quoted text does not begin a comment. This is the same comment style as upsmon.conf and many other configuration files.

Names Names are labels which identify UPS units, timers, named messages, ... They are not quoted and are made up of the 69 characters a-zA-Z0-9._%+-:@ . The leading character must be one of the 53 characters a-zA-z_ .

Numbers Numbers are non-negative and may be floating point. They are not quoted. E.g. 5.5 .

Tokens The tokens are names given to every piece of input that is recognisable by the lexer. They are shown in figure 131. The tokens are presented in the order in which they are tested by the lexer.

Quoted text Text is always quoted. The possible quotation marks are shown in figure 130. E.g. "text", 'text', «text», ⌈text⌋ and ⌜text␣. A quoted text may not contain a newline or it's terminating quote character. E.g. «te»xt» is an error as is «te
xt».

Statuses The lexer recognises the following UPS statuses: None ALARM BOOST BYPASS CAL CHRG DEAD DISCHRG FSD LB COMM OB OFF OL OVER RB TEST TICK TOCK TRIM

Events An event is a transition from one status to another, and is seen by the lexer as STATUS RARR STATUS, e.g. None->LB .

[21] See David Beazley's PLC (Python Lex-Yacc) page at https://www.dabeaz.com/ply/

Token		Use	Token		Use
1	ignore	Ignore spaces and tabs	2	newline	Line counter
3	ignore_COMMENT	Ignore #...	4	WHEN	Keyword
5	WALL	Keyword	6	USER	Keyword
7	UPSDUSER	Keyword	8	TYPE	Keyword
9	TIMEOUT	Keyword	10	SYSLOG	Keyword
11	SUBJECT	Keyword	12	STARTTIMER	Keyword
13	SMTPSERVER	Keyword	14	SHUTDOWNCMD	Keyword
15	SHELLCMD	Keyword	*Not used*		
16	SETFSD	Keyword	*Not used*		
18	REPORTS	Keyword	*Not used*		
20	RARR	Symbol ->	21	QUOTETEXT5	⌜text⌟
22	QUOTETEXT4	⌈text⌋	23	QUOTETEXT3	«text»
24	QUOTETEXT2	"text"	25	QUOTETEXT1	'text'
26	PRINT	Keyword	27	POWERVAL	Keyword
28	PORT	Keyword	29	POLLFREQALERT	Keyword
30	POLLFREQ	Keyword	31	PASSWORD	Keyword
32	NUTLOG	Keyword	33	NUMBER	0 through 9 plus .
34	NOTIFY	Keyword	*Not used*		
36	MONITOR	Keyword	37	MINSUPPLIES	Keyword
38	MESSAGE	Keyword	39	MAXNOTIFY	Keyword
40	LET	Keyword	41	IF	Keyword
Not used			43	HOST	Keyword
Not used			44	GROUP	Keyword
45	FROM	Keyword	46	EQ	Symbol =
47	EPRINT	Keyword	48	EMAIL	Keyword
49	DEBUG	Keyword	*Not used*		
51	COLON	Symbol :	52	CERTFILE	Keyword
Not used			53	CANCELTIMER	Keyword
54	APCUPSDUSER	Keyword	55	STATUS	See status list
56	TO	Keyword	57	NAME	Starts with a-zA-z_ then a-zA-Z0-9._%+-:@

Figure 131: UPSmon.conf lexer tokens.

26.2 Yacc Grammar

The grammar shows the logical structure of the configuration file. There is no separate "yacc" grammar file. The productions are represented by functions such as the one shown in figure 132.

```
1067 def p_configuration (p) :
1068   'configuration : intros groups'
1069   tag = ('configuration', p.lineno(len(p)-1)//LN, p.lineno(len(p)-1)%LN)
1070   AST = (tag, p[1], p[2])
```

Figure 132: Representation of grammar production

Line 1067 declares the function providing the grammar production seen in line 1068 for the `configuration` production. The result is tagged with a 3-tuple seen in line 1069 giving the identity, line number and column number, and forms the basis for the abstract syntax tree `AST`. The values for `p[1]` and `p[2]` in line 1070 are provided by functions `p_intros` and `p_groups`.

Production	Notes
`configuration : intros groups`	Start here
`intros : intro` `| intros intro`	Start of introduction

`intro : smtp`
`| let`
`| pollfreqalert`
`| pollfreq`	

`smtp : SMTPSERVER quotetext PORT number` `USER quotetext PASSWORD quotetext`	
`let : LET name EQ quotetexts`	`battery.charge.low.`i for $i = 1..3$ the `name` is a special value.
`number : NUMBER`	
`pollfreqalert : POLLFREQALERT number`	
`pollfreq : POLLFREQ number`	End of the introduction

continued ...

Figure 133: `UPSmon.conf` grammar.

... continued

groups	: group	Start of group specs
	| groups group_element	
group_element	: group_name	
	| group_host	
	| group_port	
	| certfile	
	| let	
	| monitors	
	| minsupplies	
	| action_declarations	
group_name	: GROUP name	
name	: NAME	
group_host	: HOST name	
group_port	: PORT number	
certfile	: CERTFILE quotetext	
	| CERTFILE name	
monitors	: monitor	
	| monitors monitor	
monitor	: MONITOR name POWERVAL number user PASSWORD quotetext TYPE name	
user	: UPSDUSER name	
	| APCUPSDUSER name	
minsupplies	: MINSUPPLIES number	
action_declarations	: action_declaration	
	| action_declarations action_declaration	
action_declaration	: event_key actions	
event_key	: WHEN name TO name COLON	TO ≡ TIMEOUT
	| WHEN name TIMEOUT name COLON	
	| WHEN name REPORTS STATUS RARR STATUS COLON	
actions	: action_element	
	| actions action_element	

continued ...

Figure 134: UPSmon.conf grammar, continued.

... continued		
action_element :	condition cancel_timer | condition debug_level | condition email | condition start_timer | condition EPRINT quotetexts | condition NOTIFY quotetexts | condition NUTLOG quotetexts | condition PRINT quotetexts | condition SETFSD name | condition SHELLCMD quotetexts | condition SHUTDOWNCMD quotetexts | condition SYSLOG quotetexts | condition WALL quotetexts	
condition :	IF STATUS RARR STATUS | empty	
quotetexts :	quotetext | name | quotetexts quotetext | quotetexts name	
quotetext :	QUOTETEXT1 | QUOTETEXT2 | QUOTETEXT3 | QUOTETEXT4 | QUOTETEXT5	
cancel_timer :	CANCELTIMER name	
debug_level :	DEBUG number	0, 1 or 2
start_timer :	STARTTIMER name number	
email :	EMAIL from to subject content	
from :	FROM quotetext	
to :	TO quotetext	
subject :	SUBJECT quotetext	
content :	MESSAGE quotetexts	
empty :		

Figure 135: UPSmon.conf grammar, final part.

26.3 Log rotation for upsdTLS.py and UPSmon.py

The well known Unix/GNU Linux utility program `logrotate` provides a convenient way of managing log files. See `man logrotate(8)`. NUT 2.7.4 already provides a declaration for it's log files. The following declaration provides separate management for the log files created by upsdTLS.py and UPSmon.py.

The file should be created as `/etc/logrotate.d/NUT` with ownership `root:root` and permissions `644`.

```
1071 # Log rotation configuration for upsdTLS.py, UPSmon.py
1072 # Rotate NUT log file either monthly or when exceeding 5 Mb
1073 #
1074 # For more information, refer to logrotate(8) manual page:
1075 #   http://linuxcommand.org/man_pages/logrotate8.html
1076 #
1077 /var/log/NUT.log {
1078     missingok
1079     notifempty
1080     size=5M
1081     rotate 12
1082     monthly
1083     create 0600 upsd root
1084 }
```

Figure 136: Log rotation for upsdTLS.py and UPSmon.py

Lie 1082 calls for a log rotation every month, and line 1081 requires keeping 12 previous months' logs, so in all there will be one year's records.

27 Acknowledgments

Editor: As one of the many who have used the work of the NUT project as part of their system setup, I would like to express my gratitude and my appreciation for the software that the NUT project has made available to system administrators through contributions by Charles Lepple, Arjen de Korte, Arnaud Quette, Jim Klimov, Russell Kroll, and many others in the nut-upsuser mailing list.

I would also like to thank those who commented on earlier versions of this text: M.B.M.

28 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. 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`