Apcupsd is a UPS control system that permits orderly shutdown of your computer in the event of a power failure.

Kern Sibbald

March 24, 2005
This manual documents apcupsd version 3.10.17
Copyright (C) 1999-2005 Kern Sibbald

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the name Apcupsd, the copyright notice, and this notice are preserved.

Apcupsd source code is released under the GNU General Public License version 2. Please see the file COPYING in the main source directory.

For more information on the project, please visit the main web site at http://www.apcupsd.com

Contents

List of Figures

List of Tables

Apcupsd Manual

Release Notes

This release contains a good number of cleanups and bug fixes to prior 3.10.x versions, and is intended to be the official release. See the ChangeLog below for more details.

New Features

```
- Implement USB on all *BSD systems. Note, the kernel
  drivers on most of these systems are still fragile.
  There are known problems, for example, on FreeBSD.
- Fix killpower on USB UPSes to properly turn off UPS.
- More killpower fixes for BackUPS Pros.
- Fix killpower sequence for serial UPSes.
```

Change Log for current version

```
----> Release 3.10.17 xxMar05
- Update default apcupsd.conf to recommend a blank DEVICE setting for USB
  driver.
- Add /dev/hiddev? to Linux USB driver device node search path.
- Add Mac OS X startup script
- Add new *BSD USB driver to support USB UPSes on FreeBSD, OpenBSD, and NetBSD.
  THIS DRIVER IS BETA SOFTWARE AND HAS A KNOWN LOCKUP ISSUE ON FREEBSD. Please
  keep this in mind when deciding whether or not to deploy it. PLEASE READ THE
  "CHECKING OUT YOUR USB SUBSYSTEM (BSD)" SECTION OF THE MANUAL as it contains
  crucial details on how to configure your system for the new driver.
- Add BackUPS Pro shutdown code to USB driver
- Prefer BackUPS style shutdown over SmartUPS in USB driver to resolve shutdown
  issues on BackUPS CS models
- Restructure USB driver to share common code
- Fix slave mode segfault bug introduced by --killpower fixes in 3.10.16.
- Commit kernstodo
- Added an anonymous patch to powerflute.c and to the slack-apcupsd.in file.
- Add Whitebox to detected systems.
- Minor tweak to RedHat spec.in
- Apply Carl Lindbergs patch
  for apcaction.c to fix the network management card
  shutdown.
- Fix typo in targets.mak that prevents uninstall from working.
- Change name of thread_terminate to apc_thread_terminate to avoid
  conflict on AIX.
- Put configure found SHUTDOWN in apccontrol.in
- Figured out how to scale the pdf images, so re-did them.
- Some minor updates to the manual, particularly the title
```

page.

Change Log for older versions

```
----> Release 3.10.16 04Nov04
- Adam has fixed the killpower problem for USB so that the
  USB now properly turns off the power. Nice job.
- Converted manual from docbook to texinfo format. There is some
  cleanup to be done, but we get an index.
- Thanks to Adam for converting the .png images to .pdf
- Apply patch to fix aastr... supplied by Manfred Schwarb.
- Changed Solaris to use mailx by default at the suggestion of
  Neil Brookins.
- Added Adam's snoopdecode.c to examples that allows viewing
  USB events.
- A number of typos fixed in apccontrol files.
- Adam fixed a race condition in killpower with --kill-on-powerfail.
- --kill-on-powerfail disallowed for dumb UPSes since the
  kill power will always occur before the system has been halted.
- Lots of doc updates.
- Add proper platform code so that configure will create
  the 4 platform specific apccontrol files (some were missing).
- Apply fix from user to correct one of the shutdown
  sequences for the Smart UPS. During the conversion to drivers
  this was apparently mangled.
- Added code to close all file descriptors before becoming
  daemon unless debug turned on.
- Add APCBATTCapBeforeRestore found by Adam to hid-ups.c
- Update copyright in apc_struct.h
- Take Adam's new apc_defines.h with minor modification.
- Correct a bug reported by a user (he also had the fix) to
  the snmp driver where Sensitivity was incorrectly reported.
- Add astrncpy() to snmp driver.
- Fix apcstatus.c to report Unknown for the sensitivity rather than
  High if the sense word cannot be read or is incorrect.

----> Release 3.10.15 07Aug04
- Document Mandrake USB kernel problems.
- Fix HID_MAX_USAGES in the examples directory
- Apply David Walser's patch for missing colors in multimon.  Reads
  the apcupsd.css file from the sysconf directory.
- Add EEPROM fix from Giuseppe Ghibo passed on by David Walser

----> Release 3.10.14 28Jul04
- Add workaround from Adam for linux/hiddev.h missing define.
- Updates to manual.
- Integrate patch for Mandrake apcupsd.in forwarded by David Walser.
- Found another store into the ups buffer so ifdefed it. Cannot
  store into the ups buffer on non-pthreads systems.
- Fiddle with apcconfig.c to correct astrncpy() problem noted by
  Adam.
```

```
- ifdef code in apcaccess that tries to write in the shared memory
  buffer.
- Applied Adam's patch for fixing the pthreads dependencies in asys.c
- Tweak the patch a bit hopefully so that OpenBSD will work.
- Made a sweep through quite a few files updating the copyright,
  eliminating the disclaimer (now in DISCLAIMER), and adding as many
  astrncpy() and astrncat()s as I could find. There still remain some
  drivers and the cgi code to do.
- Implemented true/false and bool. Started implementing it in many of
  the files I touched.
- Updated the net driver and did a fairly good testing of it.
- Made apcupsd remain in the foreground when doing a kill power.
- Eliminated some of the error messages during kill power by not
  doing useless things.
- Added back code to print what is happening during kill power
  in the USB code.
- Corrected a few of the USB error messages that could have been
  missleading or confusing.
- Eliminated some inappropriate usages of size_t.
- Integrated a number of updates into the manual, particularly from
  Adam.
- If the IP address is 0.0.0.0 force it to localhost in apcaccess.
- Integrat Thomas Habets' changes to keep connect() from blocking
  in apcnet.c so that apcupsd can service more slaves.
- Ensure that stdin/out/err are allocated in daemon_start() of apcuspd.c
- Update snmp.c with bug fix from Bacula.
- Bill has made numerous changes to improve the code such as adding
  consts where appropriate.

----> Release 3.10.13 20Apr04
- Added code to support net snmp configured with --enable-net-snmp
  based on patch sent by Sander Siemonsma.
- Build smtp on Unix systems.
- Update to most current smtp and make it easier to configure
  for apcupsd or Bacula
- Start implementing native Win32 version.
- Rename cmd - ups_event and cmd_msg - event_msg
- Add user supplied code to make apcaccess read the conf file and
  self configure to proper port. Thanks to Martin Flack for this
  patch.
- Start simplifying the Copyright and making the dates current.
- Rework the net driver. It was really in poor shape.
- Replace sprintf with asnprint. Replace strcpy with astrncpy
- Apply a fix supplied by Jim Pick where syslog releases the
  usb port and then re-attaches it to /dev/log.
- I finally took a careful look at the old master/slave networking
  code as well as ran it here, and it was sadly broken. Hopefully
  this commit fixes the problems.
- Fix a few string functions using the new routines.
- Added asys.c imported from Bacula, which contains a number of
  simple system routines such as astrncpy(), ...
```

How To Use This Manual

This is the manual for apcupsd, a daemon for communicating with UPSes (Uninterruptible Power Supplies) made by American Power Corporation (APC). If you have an APC-made UPS, whether sold under the APC nameplate or OEMed (The HP PowerTrust 2997A UPS has been tested as a "smartups" with cable Hewlett Packard part number 5061-2575 equivalent to a custom-smart cable), and you want you get it working with a computer running Linux, Unix, or Windows NT, you are reading the right document.

This manual is divided into parts which increase in technical depth as they go. If you have just bought a state-of-the-art smart UPS with a USB or Ethernet interface, and you are running a current version of Red Hat or SUSE Linux (8.0 or later), then apcupsd is very nearly plug-and-play and you will have to read only the Basic User's Guide (see Basic User's Guide).

If your operating system is older, or if you have an old-fashioned serial-line UPS, you'll have to read about serial installation (see Installation on Serial-Line UPSes). If you need more details about administration for unusual situations (such as a master/slave or multi-UPS setup) you'll need to read the section on advanced topics (see Advanced topics). Finally, there is a Technical Reference (see Technical Reference) section which gives full details on things like configuration file directives and event-logging formats.

You should begin by reading the Quick Start (see Quick Start for Beginners) instructions.

Basic User's Guide

1 Planning Your Installation

1.1 Quick Start for Beginners

apcupsd is a complex piece of software, but most of its complexities are meant for dealing with older hardware and operating systems. On current hardware and software getting it running should not be very complicated.

The following is a help guide to the steps needed to get apcupsd set up and running as painlessly as possible.

1. First, check to see if apcupsd supports your UPS and operating system (see Supported Operating Systems; UPSes and Cables).

2. Second, plan your configuration type (see Choosing a Configuration Type). If you have just one UPS and one computer, this is easy. If you have more than one machine being served by the same UPS, or more than one UPS supplying power to computers that are on the same local network, you have more choices to make.

3. Third, figure out if you have one of the easy setups. If you have a USB UPS, and a USB-capable recent Linux such as Red Hat or SuSE at version 8.0, and you want to use one UPS with one computer, that's an easy setup. APC supplies the cable needed to talk with that UPS along with the UPS. All you need to do is check that your USB subsystem is working (see Checking Out Your USB Subsystem); if so, you can go to the build and install step.

4. If you have a UPS designed to communicate via SNMP over Ethernet, that is also a relatively easy installation. It's in Advanced Topics (see Advanced topics) mainly because it's an unusual situation.

5. If you have a UPS that communicates via an RS232C serial interface and it is a SmartUPS, then things are relatively simple, otherwise, your life is about to get interesting.

   (a) If you have a vendor-supplied cable, find out what cable type you have by looking on the flat ends of the cable for a number, such as 940-0020A, stamped in the plastic. Check the cables column of the table of types (see type_table) to see if it's a supported type.

   (b) If you don't have a vendor-supplied cable, or your type is not supported, you may have to build one yourself (see Cables). Here is hoping you are good with a soldering iron!

6. Now you are ready to read the Building and Installing (see Building and Installing apcupsd) section of the manual and follow those directions. If you are installing from an RPM or some other form of binary package, this step will probably consist of executing a single command.

7. Tweak your /etc/apcupsd/apcupd.conf file as necessary. Often it will not be.

8. Change the BIOS settings (see Arranging for Reboot on Power-Up) on your computer so that boots up every time it gets power. (This is not the default on most systems.)

9. To verify that your UPS is communicating with your computer and will do the right thing when the power goes out, read and follow the instructions in the Testing (see Testing Apcupsd) section.

10. If you run into problems, read the Troubleshooting (see Troubleshooting Your Installation) section of this manual.

11. If you still need help, send a message to the developer's email list apcupsd-users at lists.sourceforge.net describing your problem, what version of apcupsd you are using, what operating system you are using, and anything else you think might be helpful.

12. Read the manual sections on monitoring and maintaining your UPS.

1.2 Supported Operating Systems, UPSes and Cables

Please note that due to the lack of Unix USB API standards, the USB code in apcupsd works only on Linux and *BSD systems. In addition, at the current release (3.10.17) the USB support for *BSD systems can at best be considered BETA due to fragile kernel drivers. Drivers for other OSes can be written, but it requires someone with a knowledge of the OS and the USB to do so. (This lack of a Unix USB API interface is one of the big failings of Unix. It occurs in other areas such as the GUI. Many people tout the diversity as an advantage, but it is in fact a weakness.)

The apcupsd maintainers develop it under Fedora (Red Hat); that port is, accordingly, the most up to date and best tested. There are enough Debian Linux users that that port is also generally pretty fresh. Slackware Linux is also fully supported.

apcupsd has also been ported to FreeBSD, NetBSD, OpenBSD, HP/UX, Solaris, Alpha Unix and the Cygwin Unix emulation under Windows. It is quite likely to work on those systems, though the port may occasionally get stale and require minor tweaking.

In Operating System Specifics you'll find operating-system-specific tips for building and configuring apcupsd.

You can generally count on your UPS being supported if it has either an Ethernet-connected SNMP interface or a USB interface with an APC-supplied cable.

The "UPSTYPE Keyword" field is the value you will put in your /etc/apcupsd/apcupd.conf file to tell apcupsd what type of UPS you have.

We'll describe the possible values here, because they're a good way to explain your UPS's single most important interface property – the kind of protocol it uses to talk with its computer.

apcsmart An APCSmart UPS and its computer also communicate through an RS232C serial connection, but they actually use it as a character channel (2400bps, 8 data bits, 1 stop bit, no parity) and pass commands back and forth in a primitive language (see APC smart protocol) resembling modem-control codes. The different APC UPSes all use closely related firmware, so the language doesn't vary much (later versions add more commands). This class of UPS is in decline, rapidly being replaced in APC's product line by USB UPSes.

usb A USB UPS speaks a universal well defined control language over a USB wire. Most of APC's lineup now uses this method as of late 2003, and it seems likely to completely take over in their low- and middle range. Other manufacturers (Belkin, Tripp-Lite, etc.) are moving the same way, though with a different control protocol for each manufacturer. As long as USB hardware can be mass-produced more cheaply than an Ethernet card, most UPSes are likely to go this design route. Please note that even if you have a USB UPS, if you use a serial cable with it (as can be supplied by APC), you will need to configure your UPS as **apcsmart** rather than **usb**.

net This is the keyword to specify if you are using your UPS in Slave mode (i.e. the machine is not directly connected to the UPS, but to another machine which is), and it is connected to the Master via an ethernet connection. You must have apcupsd's Network Information Services NIS turned on for this mode to work. It is a much simpler form of running a Slave than the old Master/Slave code.

snmp SNMP UPSes communicate via an Ethernet NIC and firmware that speaks Simple Network Management Protocol.

dumb A dumb or voltage-signaling UPS and its computer communicate through the signal lines on an RS232C serial connection. Not much can actually be conveyed this way other than an order to shut down. Voltage-signaling UPSes are obsolete; you are unlikely to encounter one other than as legacy hardware. If you have a choice, we recommend you avoid simple signalling UPSes.

The table shown below lists the APC model supported, and the possible kewords that you would use in the configuration with the listed cables. Some

of the models, particularly USB enabled models, can be run in multiple modes, so they may appear more than once in the table. APC is putting out new models at a furious rate, and so it is very likely that your model is not listed in the table. If it is USB enabled, it will probably work in USB mode. Please note that some of these new models are extremely inexpensive, so they are stripped down versions of more expensive units, and as such they do not offer as many features, so some of the example output you see elsewhere in this manual may not be available with your unit.

APC Model	*UPSTYPE Keyword*	*UPSCABLE keywords for Cables Supported*	*Status*
BackUPS CS/ES (serial mode)	apcsmart	smart (note: using Smart Custom RJ45) the new Back-UPS RS 500 models are reported NOT to work with this cable.	Supported
BackUPS Pro, Smarter Back-UPS Pro	apcsmart	940-0095A	Supported
SmartUPS, SmartUPS VS (It has not been confirmed that the cable shipped with the VS is a 940-0095.), PowerStack 450, Matrix UPS, Share-UPS Advanced Port	apcsmart	smart (note: using Smart-Custom), 940-0024C	Supported

BackUPS CS USB, Pro USB, ES USB, RS/XS 1000, RS/XS 1500, and probably other USB models	usb	usb (note: using APC cables 940-0127A/B/C)	Supported in version >=3.9.8
SmartUPS USB, Back-UPS Office USB, and any other USB UPS	usb	usb (note: using APC cable, no number)	Supported, version >=3.9.8
All SNMP-capable models	snmp	ether	Supported
BackUPS	dumb	simple (note: using Simple-Custom (This cable is not an APC product. You have to build it yourself using the instructions in Cables.), 940-0020B, 940-0020C, 940-0119A, 940-0023A	Supported
BackUPS Office, BackUPS ES	dumb	940-0119A	Supported
BackUPS CS and possibly ES models (serial mode)	dumb	940-0128A	Supported
ShareUPS Basic Port	dumb	940-0020B, 940-0020C, 940-0023A	Supported

There are three major ways of running apcupsd on your system. The first is a standalone configuration where apcupsd controls a single UPS, which

powers a single computer. This is the most common configuration. If you're working with just one machine and one UPS, skip the rest of this section.

Your choices become more interesting if you are running a small cluster or a big server farm. Under those circumstances, it may not be possible or even desirable to pair a UPS with every single machine. apcupsd supports some alternate arrangements.

The second type of configuration is a master/slave configuration, where one UPS powers several computers, each of which runs a copy of apcupsd. The computer that controls the UPS is called the master, and the other computers are called slaves. The master copy of apcupsd communicates with and controls the slaves via an Ethernet connection. This type of configuration may be appropriate for a small cluster of machines. Some example configuration files for the master and the slave machines can be found in the examples directory of the source distribution. The more recent examples are in master.apcupsd.conf and slave.apcupsd.conf.

The third configuration (new with version 3.8.3), is where a single computer controls multiple UPSes. In this case, there are several copies of apcupsd on the same computer, each controlling a different UPS. One copy of apcupsd will run in standalone mode, and the other copy or copies will normally run in master/slave mode. This type of configuration may be appropriate for large server farms that use one dedicated machine for monitoring and diagnostics

Here is a diagram that summarizes the possibilities:

Configuration types.

```
\addcontentsline{lof}{figure}{Configuration Types}\includegraphics{./main_configs.eps}
```

If you decide to set up one of these more complex configurations, see the Advanced Topics (see Advanced topics) section for details.

1.3 Apcupsd Known USB Issues

- **Problem:** USB is only supported on Linux and *BSD systems (though the *BSD is still BETA). Although the configuration script allows the usb

driver to be enabled on other platforms, it will only compile and run on Linux and *BSD systems.

- **Workaround:** Try using UPS in serial mode instead of USB.

- **Problem:** Linux 2.4 series kernels older than 2.4.22 do not bind the USB device to the proper driver. This is evidenced by /proc/bus/usb/devices listing the UPS correctly but it will have "driver=(none)" instead of "driver=(hid)". This affects RHEL3, among others.

- **Workaround:** Upgrade linux kernel to 2.4.22 or higher.

- **Problem:** Mandrake 10.0 and 10.1 systems with high security mode enabled (running kernel-secure kernel) use static device nodes but still assign USB minor numbers dynamically. This is evidenced by **hiddev0: USB HID v1.10 Device [...]** instead of **hiddev96: ...** in dmesg log.

- **Workaround:** Boot standard kernel instead of kernel-secure or disable CONFIG_USB_DYNAMIC_MINORS and rebuild kernel-secure.

- **Problem:** USB driver linux-usb.c fails to compile, reporting errors about **HID_MAX_USAGES undefined**. This is due to a defect in the linux kernel hiddev.h header file on 2.6.5 and higher kernels.

- **Workaround:** Workaround: Upgrade to apcupsd-3.10.14 or higher. These versions contain a workaround for the defect.

- **Problem:** On some systems such as Slackware 10.0, no USB devices will showup (see the next section).

- **Workaround:** add the following to rc.local

```
mount -t usbdevfs none /proc/bus/usb
```

- **Problem:** 2.6 kernels use udev and does not autmatically create /dev/usb/hiddev?? as it should, causing apcupsd to

- **Workaround:** Edit the file /etc/udev/rules.d/50-udev.rules, and add the following:

```
KERNEL="hiddev*", NAME="usb/hiddev%n"
```

More details are provided in the following section ...

1.4 Checking Out Your USB Subsystem

You can skip this section if your UPS has an Ethernet or RS232-C interface or you are not running on a Linux kernel. If it has a USB interface, you need to make sure that your USB subsystem can see the UPS. On a Linux system this is easy, just do this from a shell prompt (please see below for 2.6 kernel considerations):

Most of this section applies to Linux. However, toward the end, there is a list of known issues with the BSD USB driver and the associated kernels.

```
cat /proc/bus/usb/devices
```

This information is updated by the kernel whenever a device is plugged in or unplugged, irrespective of whether apcupsd is running or not. To interpret the codes in this file, please see http://www.linuxhq.com/kernel/v2.4/doc/usb/proc_usb_info.txt.html

You should get some output back that includes something like this from ESR's site, featuring an RS 1000:

```
T:  Bus=02 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  3 Spd=1.5 MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=051d ProdID=0002 Rev= 1.06
S:  Manufacturer=American Power Conversion
S:  Product=Back-UPS RS 1000 FW:7.g3 .D USB FW:g3
S:  SerialNumber=JB0308036505
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr= 24mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=00 Prot=00 Driver=hid
```

Note, if on the last line, **Driver** is listed as **Driver=none** then you do not have the HID driver loaded or the driver did not attach to the UPS. One common cause is having a Linux kernel older than 2.4.22 (such as a stock RedHat 9 kernel). If this is the case for your system, please upgrade to at least kernel version 2.4.22 and try again. Otherwise, please read further for instructions for other possible courses of action.

For more details on how to interpret these codes, please see the end of this section.

Here are two more ample entries from Kern Sibbald. The first features a Back-UPS 350 direct connected USB device:

```
T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=1.5 MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=051d ProdID=0002 Rev= 1.00
S:  Manufacturer=American Power Conversion
S:  Product=Back-UPS 350 FW: 5.2.I USB FW: c1
S:  SerialNumber=BB0115017954
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr= 30mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=00 Prot=00 Driver=hid
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl= 10ms
```

The second features an IOgear USB-to-serial adapter that runs my serial SmartUPS 1000:

```
T:  Bus=01 Lev=01 Prnt=01 Port=01 Cnt=02 Dev#=  4 Spd=12  MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0557 ProdID=2008 Rev= 0.01
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 3 Cls=ff(vend.) Sub=00 Prot=00 Driver=serial
E:  Ad=81(I) Atr=03(Int.) MxPS=  10 Ivl=  1ms
E:  Ad=02(O) Atr=02(Bulk) MxPS=  64 Ivl=  0ms
E:  Ad=83(I) Atr=02(Bulk) MxPS=  64 Ivl=  0ms
```

Note that the IOgear device is using the **serial** driver (the I: line) while the Back-UPS 350 is using the **hid** driver.

In general, if you see your UPS model in the S field, which means **Manufacturer=**, **Product=**, and **SerialNumber=**, and you see **hid** in the I field (or **serial** if you are using an IOGear connection), you're done. You can skip the rest of this section and go straight to building and installing.

If it doesn't show, check the obvious things; the UPS must be powered on, and a cable must be properly seated in both the data port of the UPS and one of your machine's USB ports. Many UPSes have phone ports to provide surge protection for phones or modems – make sure you haven't plugged your USB cable into one of those rather than the data port (which will usually be near the top edge of the case.)

Note, on recent Debian systems, they do not include the hiddev device nodes in /dev, so you may need to manually create them using the **examples/make-hiddev** script.

Also, ensure that the correct drivers are loaded. Under Linux-2.4.x, you can check this out easily by examining the right file in the /proc system. Here's how you can do that:

```
esr@grelber$ cat /proc/bus/usb/drivers
```

and you should get:

```
        usbdevfs
        hub
 96-111: hiddev
        hid
```

On Linux-2.6.x, make sure the sysfs filesystem is mounted on /sys and do:

```
adk0212@mail$ ls -l /sys/bus/usb/drivers/
```

where you should get

```
total 0
drwxr-xr-x    2 root     root            0 May  1 18:55 hid
drwxr-xr-x    2 root     root            0 May  1 18:55 hiddev
drwxr-xr-x    2 root     root            0 May  1 18:55 hub
drwxr-xr-x    2 root     root            0 May  1 18:55 usb
drwxr-xr-x    2 root     root            0 May  1 18:55 usbfs
```

or perhaps something like

```
total 0
drwxr-xr-x  2 root root 0 Jan  6 15:27 hiddev
drwxr-xr-x  2 root root 0 Jan  6 15:28 hub
drwxr-xr-x  2 root root 0 Jan  6 15:28 usb
drwxr-xr-x  2 root root 0 Jan  6 15:27 usbfs
drwxr-xr-x  2 root root 0 Jan  6 15:28 usbhid
```

If your 2.6.x system does not have the /sys/bus/usb directory, either you do not have sysfs mounted on /sys or the USB module(s) have not been loaded. (Check /proc/mounts to make sure sysfs is mounted.)

A USB UPS needs all of these drivers – the USB device filesystem, the USB hub, the Human Interface Device subsystem driver, and the Human Interface Device driver. If you are compiling your own kernel, you want to enable

CONFIG_USB, CONFIG_USB_HID, CONFIG_USB_HIDDEV, and CONFIG_USB_DEVICEFS as well as at least one USB Host Controller Driver (CONFIG_USB_UHCI_HCD [2.6.x], CONFIG_USB_UHCI [2.4.x], etc.).

If CONFIG_USB is set as M, CONFIG_USB_HID must be M (if enabled at all). If CONFIG_USB is set as Y, CONFIG_USB_HID can be M or Y. hiddev, in turn, will be built however HID is.

To complicate things more many Linux flavors running 2.6 kernels such as Fedora FC3 use the udev filesystem, which creates the devices in /dev (as well as some others such as network devices) on the fly as they are needed. It is basically a hotplug system, giving a lot more power to the user to determine what happens when a device is probed or opened. It is also a lot more complicated.

The bottom line for apcupsd on such a system is that the file /dev/usb/hiddev# is not defined, and hence apcupsd terminates in error. The solution to the problem is to add a rule to the udev rules file. On Fedora FC3, this file is found in /etc/udev/rules.d/50-udev.rules. Start by adding the following line:

```
BUS="usb", SYSFS[idVendor]="051d", NAME="usb/hiddev%n"
```

where you replace the [and] with braces in the line above.

Then either reboot your system, or unplug and replug your UPS and then restart apcupsd. At that point apcupsd should work fine. You can use:

```
udevinfo -a -p /sys/class/usb/hiddev0/
```

to get more information on the fields that can be matched.

Adam has provided the following as a more generic rule:

```
KERNEL="hiddev*", NAME="usb/hiddev%n"
```

If you have several UPSes or you just want to give your UPS a fixed name, you can use:

```
BUS="usb", SYSFS[serial]="AS0123456789", NAME="usb/ups0"
```

where you replace the [and] with braces and the serial number with the one that correspnds to your UPS.

Some kernels ship, such as Mandrake 10, ship with CONFIG_USB_DYNAMIC_MINORS turned on. This is not ideal for running with apcupsd, and the easiest solution is to turn CONFIG_USB_DYNAMIC_MINORS off and rebuild your kernel, or find a pre-built kernel with it off. For a kernel with CONFIG_USB_DYNAMIC_MINORS turned on to work with apcupsd, you must enable **devfs**. The following will tell you if devfs is enabled:

```
$ ps ax | grep devs
```

which should give something like the following:

```
533 ?        S      0:00 devfsd /dev
```

What complicates the situation much more on Mandrake kernels is their security level since CONFIG_DYNAMIC_USB_MINORS is turned on, but on higher security levels devfs is turned off. The net result, is that in those situations hiddev is hosed (to use Adam's terms) so apcupsd will not work. So, in these cases, the choices are:

```
(a) Reduce the security level setting of the system
    (not sure if this is possible after the initial install).
(b) Custom build a high security kernel with devfs enabled
    and make sure devfs is mounted and devfsd is running.
(c) Custom build a high security kernel with dynamic
    minors disabled
(d) Use udev
```

For a typical USB section of a kernel .config file, please see the end of this section.

For the IOGear serial USB connection, you need:

```
usbcore
usbserial
pl2303
```

Finally, check that appropriate USB devices exist. On a Red Hat system you can do this:

```
esr@grelber$ ls /dev/usb/h*
/dev/usb/hiddev0   /dev/usb/hiddev12  /dev/usb/hiddev2  /dev/usb/hiddev6
/dev/usb/hiddev1   /dev/usb/hiddev13  /dev/usb/hiddev3  /dev/usb/hiddev7
/dev/usb/hiddev10  /dev/usb/hiddev14  /dev/usb/hiddev4  /dev/usb/hiddev8
/dev/usb/hiddev11  /dev/usb/hiddev15  /dev/usb/hiddev5  /dev/usb/hiddev9
```

This will tell you that the Human Interface Device nodes, one of which apcupsd will use to talk with the UPS, exist. On other Linuxes the layout will be slightly different; the hiddev devices will usually live in a /dev/usb/hid/ subdirectory. If these devices don't exist, you may need to run <apcupsd-source>/examples/make-hiddev to create them.

Now build and run the hid-ups test program. You do not have to configure and build the rest of apcupsd to do this. To build hid-ups enter:

```
cd <apcupsd-source>/examples
make hid-ups
```

There should be no errors. Now assuming that everything has gone well to this point and that you have connected your USB UPS, enter:

```
./hid-ups
```

It should print a sample report of the information that it has obtained from your UPS. CAUTION! if you have a 2.4.x Linux kernel do not run two copies of this program at the same time, or your kernel will freeze. The report that is printed should look very similar to the report in <apcupsd-source>/examples/hid-ups.rpt. If the program reports that the device was not found ensure that all the appropriate modules are loaded (as described earlier), then unplug your UPS and plug it back in. This should permit the kernel to recognize the UPS.

If ./hid-ups tells you "No permission, try this as root", you know what to try. If it says "Couldn't find USB UPS device, check your /dev.", then it is very unlikely that apcupsd will work. You probably need to run the script "make-hiddev" before continuing.

If all there things check out and you still can't see the UPS, something is more seriously wrong than this manual can cover – find expert help. If you are unable to list USB devices or drivers, you kernel may not be USB-capable and that needs to be fixed. Please check if your kernel has the three patches listed in the <apcupsd-source>/examples directory. Each of the files ends with the name .patch, and at the current writing they are:

```
linux-2.4.20-killpower.patch
linux-2.4.20-USB-reject.patch
linux-2.6.0-USB-queue-overflow.patch
```

For example, RedHat 9 and/or pre-2.4.22 kernels are known to need the linux-2.4.20-USB-reject.patch for APC SmartUPS XL series devices.

There are also a few email files that you can consult in the examples directory for additional information and details.

Finally, check your Kernel Config. You will find more information about it at:

Kernel Config.

KNOWN ISSUES WITH BSD USB

The BSD USB driver for apcupsd is BETA software and has some known issues.

- FreeBSD lockups: Some users have experienced lockups (apcupsd stops responding) on FreeBSD systems. In at least one case this problem was worked around by disabling pthreads (—disable-pthreads flag to configure). The problem seems to be caused by a FreeBSD kernel bug.

- FreeBSD kernel panics if USB cable is unplugged while apcupsd is running. This is another kernel bug and is most easily worked around by not hot-unplugging the UPS while apcupsd is running.

PLATFORMS & VERSIONS

The new (beta) FreeBSD USB driver supports FreeBSD, OpenBSD and NetBSD. (Thanks go to the *BSD developers who kept a nearly identical interface across all three platforms.)

The driver has been tested with the following platform versions:

```
FreeBSD-5.3     (Primary development platform)
```

```
FreeBSD-4.11
NetBSD-2.0
NetBSD-1.6.2
OpenBSD-3.6
```

FreeBSD-5.3 has had the most testing since it is the primary platform on which the driver is developed. The other platforms and versions have had somewhat less testing. The only architecture tested so far (on any platform) is i386, althought there is no reason to think it will not work on other archs. If you run the driver on a new platform version or architecture, please report your experience to the apcupsd-users mailing list.

KERNEL CONFIGURATION

You will need to rebuild your kernel in order to disable the uhid driver. uhid is not sufficient for apcupsd at this time and we need to prevent it from grabbing the UPS device. You should disable the following devices in your kernel config file (comment them out):

FreeBSD (you WILL NOT lose use of USB keyboard and mouse): uhid

NetBSD (you WILL lose use of USB keyboard and mouse): uhidev, ums, wsmouse, ukbd, wskbd, uhid

OpenBSD (you WILL lose use of USB keyboard and mouse): uhidev, ums, wsmouse, ukbd, wskbd, uhid

For detailed information on rebuilding your kernel, consult these references:

```
FreeBSD:
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig.html

NetBSD:
http://www.netbsd.org/guide/en/chap-kernel.html

OpenBSD:
http://www.openbsd.org/faq/faq5.html#Building
```

CHECKING UPS IS RECOGNIZED BY THE KERNEL

After building a properly configured kernel, reboot into that kernel and plug in your UPS USB cable. You should see a dmesg log message like the following:

ugen0: American Power Conversion Back-UPS RS 1500 FW:8.g6 .D USB FW:g6, rev 1.10/1.06, addr 2

Note that the "ugen" driver is called out. If you see "uhid" instead, it probably means you did not properly disable the uhid driver when you compiled your kernel or perhaps you're not running the new kernel.

You can also check with 'usbdevs -d' to get a list of USB devices recognized by the system as well as the drivers they are associated with. For example:

```
# usbdevs -d
addr 1: UHCI root hub, VIA
  uhub0
 addr 2: Back-UPS RS 1500 FW:8.g6 .D USB FW:g6, American Power Conversion
   ugen0
```

MAKING DEVICE NODES

Apcupsd communicates with the UPS through the USB generic device, ugen. You may or may not need to manually make ugen device nodes in /dev, depending on what OS you are using.

FreeBSD: No manual intervention needed. FreeBSD automatically creates the ugen nodes on demand.

NetBSD: By default, NetBSD only creates nodes for the first ugen device, ugen0. Check 'usbdevs -d' to see which device your UPS was bound to and then create the appropriate nodes by running 'cd /dev ; ./MAKEDEV ugenN', where ugenN is the ugen device name shown by usbdevs. It is probably a good idea to create several sets of ugen nodes in case you add more USB devices.

OpenBSD: Similar to NetBSD, OpenBSD creates nodes for ugen0 and ugen1. Check 'usbdevs -d' to see which device your UPS was bound to and then create the appropriate nodes by running 'cd /dev ; ./MAKEDEV ugenN', where ugenN is the ugen device name shown by usbdevs. It is probably a good idea to create several sets of ugen nodes in case you add more USB devices.

APCUPSD CONFIGURATION

Apcupsd must be built with USB support, which is accomplished via the —enable-usb switch to configure.

Your apcupsd.conf file needs the following hardware-related settings:

```
UPSCABLE usb
UPSTYPE usb
DEVICE
```

The DEVICE setting is blank on purpose; apcupsd will automatically locate your UPS.

The delay-, timeout-, and NIS-related settings should be configured as per your usual preference.

2 Building and Installing apcupsd

2.1 Installation from Binary Packages

2.1.1 Red Hat Linux: For Red Hat systems, apcupsd is available in binary RPM format. This is the simplest way to install. If you have no previous version of apcupsd on your machine and are creating a standalone configuration, simply install the RPM with a normal `rpm -ihv` command. You're done, and can now skip the rest of this chapter and go straight to tweaking your run-time configuration file. (see After Installation)

If you have a previous installation, you can upgrade with a normal `rpm -Uhv`, but this may not upgrade the halt script. It may be better to do the upgrade as a remove (`rpm -e`) foll;owed by a fresh install (`rpm -ihv`).

After installation of the binary RPM, please verify carefully that /etc/rc.d/init.d/halt was properly updated and contains new script lines flagged with *****APCUPSD*****.

Since there is no standard location for cgi-bin, the rpm will place the binary CGI programs in the directory /etc/apcupsd/cgi. To actually use them, you must copy or move them to your actual cgi-bin directory, which on many systems is located in /home/httpd/cgi-bin.

2.1.2 Microsoft Windows: If you have a binary release of the Win32 apcupsd, please see the instructions in the Advanced Topics (see Advanced topics) section of this manual.

2.2 Installation from Source

Installation from source might have to be be done different ways depending on what system you are running. The basic procedure involves getting a source distribution, running the configuration, rebuilding, and installing.

The basic installation from a tar source file is rather simple:

1. Unpack the source code from its tar archive.
2. Go into the directory containing the source code.
3. Run `./configure` (with appropriate options as described below)
4. make
5. su (i.e. become root)
6. Stop any running instance of apcupsd. The command to do this will look like `<system-dependent-path>/apcupsd stop`
7. uninstall any old apcupsd This is important since the default install locations may have changed.
8. make install
9. edit your /etc/apcupsd/apcupsd.conf file if necessary
10. ensure that your halt script is properly updated
11. Start the new apcupsd with: `<system-dependent-path>/apcupsd start`

If all goes well, the `./configure` will correctly determine which operating system you are running and configure the source code appropriately. `configure` currently recognizes the systems listed below in the Operating System Specifics section of this chapter and adapts the configuration appropriately. Check that the configuration report printed at the end of the `configure` process corresponds to your choice of directories, options, and that it has correctly detected your operating system. If not, redo the `configure` with the appropriate options until your configuration is correct.

Please note that a number of the `configure` options preset apcupsd.conf directive values in an attempt to automatically adapt apcupsd as best possible to your system. You can change the values in apcupsd.conf at a later time without redoing the configuration process by simply editing the apcupsd.conf file.

Other configuration options can be used to set up the installation of HTML documentation and optional modules, notably the CGI interface that enables the UPS state to be queried via the Web and the optional powerflute curses-based control panel. Still others enable features such as thread support. You will find a complete reference later in this chapter.

In general, you will probably want to supply a more elaborate `configure` statement to ensure that the modules you want are built and that everything is placed into the correct directories.

On Red Hat, a fairly typical configuration command would look like the following:

```
CFLAGS="-g -O2" LDFLAGS="-g" ./configure \
  --enable-usb \
  --with-upstype=usb \
  --with-upscable=usb \
  --prefix=/usr \
  --sbindir=/sbin \
  --with-cgi-bin=/var/www/cgi-bin \
  --enable-cgi \
  --with-css-dir=/var/www/docs/css \
  --with-log-dir=/etc/apcupsd \
  --enable-pthreads \
  --enable-powerflute
```

By default, `make install` will install the executable files in /sbin, the manuals in /usr/man, and the configuration and script files in /etc/apcupsd. In addition, if your system is recognized, certain files such as the startup script and the system halt script will be placed in appropriate system directories (usually subdirectories of /etc/rc.d).

2.3 Verifying a Source Installation

There are a number of things that you can do to check if the installation (make install) went well. The fist is to check where the system has installed apcupsd using `which` and `whereis`. On my Red Hat system, you should get the following (lines preceded with a $ indicate what you type):

```
$ which apcupsd
/sbin/apcupsd
$ whereis apcupsd
apcupsd: /sbin/apcupsd /etc/apcupsd /etc/apcupsd.conf
/etc/apcupsd.status /usr/man/man8/apcupsd.8.gz
/usr/man/man8/apcupsd.8
```

If you find an apcupsd in /usr/sbin, /usr/local/sbin, /usr/lib, or another such directory, it is probably a piece of an old version of apcupsd that you can delete. If you are in doubt, delete it, then rerun the `make install` to

ensure that you haven't deleted anything needed by the new apcupsd. Please note that the files specified above assume the default installation locations.

As a final check that the `make install` went well, you should check your halt script (in /etc/rc.d on SUSE systems, and in /etc/rc.d/init.d on Red Hat systems) to see that the appropriate lines have been inserted in the correct place. Modification of the halt script is important so that at the end of the shutdown procedure, apcupsd will be called again to command the UPS to turn off the power. This should only be done in a power failure situation as indicated by the presence of the /etc/powerfail file, and is necessary if you want your machine to automatically be restarted when the power returns. On a Red Hat system, the lines containing the `# ***apcupsd***` should be inserted just before the final halt command:

```
# Remount read only anything that's left mounted.
#echo "Remounting remaining filesystems (if any) readonly"
mount | awk '/ext2/ { print $3 }' | while read line; do
    mount -n -o ro,remount $line
done

# See if this is a powerfail situation.                                    # ***apcupsd***
if [ -f /etc/apcupsd/powerfail ]; then                                     # ***apcupsd***
   echo                                                                    # ***apcupsd***
   echo "APCUPSD will now power off the UPS"                               # ***apcupsd***
   echo                                                                    # ***apcupsd***
   /etc/apcupsd/apccontrol killpower                                       # ***apcupsd***
   echo                                                                    # ***apcupsd***
   echo "Please ensure that the UPS has powered off before rebooting"      # ***apcupsd***
   echo "Otherwise, the UPS may cut the power during the reboot!!!"        # ***apcupsd***
   echo                                                                    # ***apcupsd***
fi                                                                         # ***apcupsd***

# Now halt or reboot.
echo "$message"
if [ -f /fastboot ]; then
 echo "On the next boot fsck will be skipped."
elif [ -f /forcefsck ]; then
 echo "On the next boot fsck will be forced."
fi
```

The purpose of modifying the system halt files is so that apcupsd will be recalled after the system is in a stable state. At that point, apcupsd will instruct the UPS to shut off the power. This is necessary if you wish your system to automatically reboot when the mains power is restored. If you prefer to manually reboot your system, you can skip this final system dependent installation step by specifying the `disable-install-distdir` option on the `./configure` command (see below for more details).

The above pertains to Red Hat systems only. There are significant differences in the procedures on each system, as well as the location of the halt script. Also, the information that is inserted in your halt script varies from system to system. Other systems such as Solaris require you the make the changes manually, which has the advantage that you won't have any unpleasant surprises in your halt script should things go wrong. Please consult the specific system dependent README files for more details.

Please note that if you install from RPMs for a slave machine, you will need to remove the changes that the RPM install script made (similar to what is noted above) to the halt script. This is because on a slave machine there is no connection to the UPS, so there is no need to attempt to power off the UPS. That will be done by the master.

2.4 Configure Options

All the available `configure` options can be printed by entering:

```
./configure --help
```

When specifying options for `./configure`, if in doubt, don't put anything, since normally the configuration process will determine the proper settings for your system. The advantage of these options is that it permits you to customize your version of apcupsd. If you save the `./configure` command that you use to create apcupsd, you can quickly reset the same customization in the next version of apcupsd by simply re-using the same `./configure` command.

The following command line options are available for `configure` to customize your installation.

—**prefix=<path>** This defines the directory for the non-executable files such as the manuals. The default is **/usr**.

—**sbindir=<path>** This defines the directory for the executable files such as apcupsd. The default is /sbin. You may be tempted to place the executable files in /usr/sbin or /usr/local/sbin. Please use caution here as these directories may be unmounted during a shutdown and thus may prevent the `halt` script from calling apcupsd to turn off the UPS power. Though your data will be protected, in this case, your system will probably not be automatically rebooted when the power returns.

—**enable-powerflute** This option enables the building of the powerflute executable, which is a ncurses based program to monitor the UPS. This program is not necessary for the proper execution of apcupsd.

—**enable-cgi** This enables the building of the CGI programs that permit Web browser access to apcupsd data. This option is not necessary for the proper execution of apcupsd.

—**with-cgi-bin=<path>** The with-cgi-bin configuration option allows you to define the directory where the CGI programs will be installed. The default is /etc/apcupsd, which is probably not what you want.

—**with-css-dir=<path>** This option allows you to specify where you want apcupsd to put the Cascading Style Sheet that goes with the multimoncss.cgi CGI program.

—**enable-master-slave** Turns on the master/slave networking code (default). This is sometimes referred to as the old master/slave code, and is more complicated than using NIS and the net driver to control Slaves (see below).

—**enable-apcsmart** Turns on generation of the APC Smart driver (default).

—**enable-dumb** Turns on generation of the dumb signalling driver code (default).

—**enable-usb** Turns on generation of the Linux (only) USB driver code. By default this is disabled.

—**enable-net** Turns on generation of the NIS network driver for slaves. This is an alternative to old master/slave code. For the master, this code should be disabled. For each slave, this is the only driver needed. This driver works by reading the information from the the configured master using the NIS (Network Information Services) interface.

—**enable-snmp** Turns on generation of the SNMP driver. This driver will control the computer by reading the UPS information over the network assuming you are running SNMP. By default this is disabled.

—**enable-test** This turns on a test driver that is used only for debugging. By default it is disabled.

—**enable-nis** Turns on the Network Information Server (NIS) code within apcupsd. This is enabled by default. If you do not want to access the status of the UPS from the network and you are not controlling any slaves via NIS (enable-net), this can be disabled.

—**enable-pthreads** This option enables pthreads support causing apcupsd to be built as a threaded program rather than forking to create separate processes. apcupsd built in this fashion is more efficient that the standard version being one third the data size and less overhead locking and coping shared memory. This option is *highly* recommended for Windows builds.

—**with-libwrap=<path>** This option when enabled causes apcupsd to be built with the TCP WRAPPER library for enhanced security. In most cases, the <path> is optional since configure will determine where the libraries are on most systems.

—**with-nologin=<path>** This option allows you to specify where apcupsd will create the nologin file when logins are prohibited. The default is /etc

—**with-pid-dir=<path>** This option allows you to specify where apcupsd will create the process id (PID) file to prevent multiple copies from running. The default is system dependent but usually /var/run.

—**with-log-dir=<path>** This option allows you to specify where apcupsd will create the EVENTS and STATUS log files. The default is /etc/apcupsd. This option simply sets the default of the appropriate path in the apcupsd.conf file, which can be changed at any later time.

—**with-lock-dir=<path>** This option allows you to specify where apcupsd will create the serial port lock file. The default is systemdependent but usually /var/lock. This option simply sets the appropriate path in the apcupsd.conf file, which can be changed at any later time.

—**with-pwrfail-dir=<path>** This option allows you to specify where apcupsd will create the powerfail file when a power failure occurs. The default is system dependent but usually /etc.

—**with-serial-dev=<device-name>** This option allows you to specify where apcupsd will look for the serial device that talks to the UPS. The default is system dependent, but often /dev/ttyS0. This option simply sets the appropriate device name in the apcupsd.conf file, which can be changed at any later time.

—**with-nis-port=<port>** This option allows you to specify what port apcupsd will use for the Network Information Server (the CGI programs). The default is system dependent but usually 3551 because that port has been officially assigned to apcupsd by the IANA. This option simply sets the appropriate port in the apcupsd.conf file, which can be changed at any later time.

—**with-nisip=<IP-Address>** This option allows you to specify the value that will be placed on then NISIP directive in the configuration file. The default is 0.0.0.0. No checking is done on the value entered, so you must ensure that it is a valid IP address.

—**with-net-port=<port>** This option allows you to specify what port apcupsd will use for Master and Slave communications. The default is system dependent but usually 6666. This option simply sets the appropriate port in the apcupsd.conf file, which can be changed at any later time.

—**with-upstype=<type>** This option allows you to specify the type of UPS that will be connected to your computer. The default is: smartups. This option simply sets the appropriate UPS type in the apcupsd.conf file, which can be changed at any later time.

—**with-upscable=<path>** This option allows you to specify what cable you are using to connect to the UPS. The default is: smart. This option simply sets the appropriate UPS cable in the apcupsd.conf file, which can be changed at any later time.

—**disable-install-distdir** This option modifies the apcupsd Makefiles disable installation of the distribution (platform) directory. Generally, this used to do a full installation of apcupsd except the final modification of the operating system files (normally /etc/rc.d/halt, etc.). This is useful if your operating system is not directly supported by apcupsd or if you want to run two copies of apcupsd on the same system. This option can also be used by those of you who prefer to manually reboot your system after a power failure or who do not want to modify your system halt files.

2.5 Recommended Options for most Systems

For most systems, we recommend the following options:

```
./configure --prefix=/usr --sbindir=/sbin --enable-usb \
            --enable-pthreads
```

and you can optionally build and install the CGI programs as follows:

```
./configure --prefix=/usr --sbindir=/sbin --enable-usb \
            --enable-cgi --with-cgi-bin=/home/httpd/cgi-bin \
            --enable-pthreads
```

2.6 Compilers and Options

Some systems require unusual options for compilation or linking that the `./configure` script does not know about. You can specify initial values for variables by setting them in the environment. Using a Bourne-compatible shell, you can do that on the command line like this:

```
CFLAGS="-O2 -Wall" LDFLAGS= ./configure
```

Or on systems that have the `env` program, you can do it like this:

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

Or for example on the Sun Solaris system, you can use:

```
setenv CFLAGS -O2
setenv LDFLAGS -O
./configure
```

You can get a listing of all available options by doing:

```
./configure --help
```

or simply see the previous section of this manual.

2.7 Operating System Specifics

With the exception of Linux SUSE and Linux Red Hat systems used by the developers, we rely on users to help create installation scripts and instructions as well as to test that apcupsd runs correctly on their system.

As you can imagine, most of these people are system administrators rather than developers so they are very busy and don't always have time to test the latest releases. With that in mind, we believe that you will find that a lot of very valuable work has been already done to make your installation much easier (and probably totally automatic).

Below, you will find a list of operating systems for which we have received installation files:

- Alpha (see Alpha)
- Debian (see Debian)
- FreeBSD (see FreeBSD)
- HPUX (see HPUX)
- NetBSD (see NetBSD)
- OpenBSD (see OpenBSD)
- Red Hat (see Red Hat Systems)
- Slackware (see Slackware)
- SUSE (see SUSE)
- Solaris (see Sun Solaris)
- unknown (see Unknown System)
- Win32 (see Windows Systems with CYGWIN Installed)

2.7.1 Alpha: The Alpha V4.0 version of apcupsd builds without compiler errors with gcc version 2.95.2. It is unlikely that the native Alpha compiler will work because of varargs differences. Unless you are a system guru, we recommend that you connect your UPS to the second serial port /dev/tty01 to avoid conflicts with the console device.

```
DEVICE /dev/tty01
```

In addition, you should ensure serial port lock file in apcupsd.conf is defined as:

```
LOCKFILE /var/spool/locks
```

Unlike the Linux systems, the system halt routine is located in /sbin/rc0, so after the `make install`, please check that this file has been correctly updated.

The start/stop script can be found in:

```
/sbin/init.d/apcupsd
```

2.7.2 Debian: This port is complete and is operation by several users. Since Debian build and install procedures are somewhat particular, we have put the extra Debian information into the following two subdirectories: <src>/distributions/debian/examples/ and <src>/distributions/debian/packageinfo

You can also find the official Debian packages on the Debian site at:

- http://packages.debian.org/stable/admin/apcupsd.html
- http://packages.debian.org/testing/admin/apcupsd.html
- http://packages.debian.org/unstable/admin/apcupsd.html

2.7.3 FreeBSD: This port is complete and is being used by several users. As of version 3.8.3, we do not recommend that you compile apcupsd with pthreads enabled. This is because the current FreeBSD implementation of pthreads runs as a single process, and thus is less efficient (consumes more CPU time) than the forking version of apcupsd. We hope to rectify this in a future version by using the FreeBSD LinuxThreads implementation of pthreads.

On the FreeBSD OS, there is no known way for a user program to get control when all the disks are synced. This is needed for apcupsd to be able to issue the killpower command to the UPS so that the UPS shuts off the power. To accomplish the same thing on FreeBSD systems, make sure you have a SmartUPS and that your UPS shutdown grace period is set sufficiently long so that you system will power down (usually 2 minutes), the use the `kill-on-powerfail` option on the apcupsd command line.

Please note the concerns listed below under OpenBSD concerning the use of pthreads.

2.7.4 HPUX: We have no reports from testing this yet on version 3.8.4, but worked fine on 3.8.1

2.7.5 NetBSD: Submitted during development of 3.8.2, this should be a complete distribution. Please read the comments on the pthreads implementation in the FreeBSD section above as they may apply equally to OpenBSD.

Please note the concerns listed below under OpenBSD concerning the use of pthreads.

2.7.6 OpenBSD: Ensure that you read the distributions/openbsd/README file before running apcupsd. There are some critical differences in how the OpenBSD implementation operates when the UPS batteries are exhausted. Failure to take this into account may result in the system not being fully halted when power is lost. Please read the comments on the pthreads implementation in the FreeBSD section above as they may apply equally to OpenBSD.

PLEASE NOTE. Due to some deficiencies or errors in the OpenBSD pthreads libraries, if you build apcupsd on OpenBSD with pthread and a child program is launched (i.e. mail notification of events), this may cause OpenBSD to freeze up. The best solution is probably to build without pthread. However, in doing so, you must realize that the bulk of this manual assumes that pthreads is enabled, and thus many of the comments about apcaccess will not be applicable. A second solution that seems to work is to delete all calls to the email notification routines from apccontrol. In doing so, some users have succeeded in running apcupsd with pthreads.

If you want to know the technical problems with pthreads on OpenBSD, it is as best we can tell because the pthreads are not real kernel pthreads as on Linux and Solaris, but rather a user program that makes all I/O non-blocking. So when apcupds does I/O, the userland pthreads libarary will switch to another thread if it wants to run. This works fine except that when a child process is called and it exits, all the blocking/non-blocking statuses of the open file descriptors in the parent program are reset as blocking — this causes chaos and an almost immediate freezing of the program (apcupsd).

2.7.7 Red Hat Systems: Red Hat systems are fully supported, and by following the standard installation instructions given above, you should experience few or no problems.

2.7.8 Slackware: Slackware systems are fully supported, and by following the standard installation instructions given above, you should experience few or no problems.

2.7.9 SUSE: SUSE systems are fully supported, and by following the standard installation instructions given above, you should experience few or no problems.

2.7.10 Sun Solaris: Please read this before attempting to compile or install the beta software. It contains important information that will make your efforts easier.

If you find bugs, or run into problems that seem to be related to the version of Solaris that you run, please feel free to contact the maintainers by email, or through the development mailing list. We'll attempt to help with problems getting the beta running, although we can't promise a quick response.

As always, remember testing UPSes can be hazardous to you system, and, apcupsd *may contain bugs that can damage your system and data files!* You must accept all responsibility for running this software. An unexpected power-off of a running system can be a disaster. As always, make backups of any critical information before you install this software.

Remember, we told you. we'll listen sympathetically if you lose data, but there will be nothing we can do to help you.

Please read the general installation instructions given above before continuing on with these Solaris-specific instructions. Then come back and read this section before attempting to build the package.

For building the system, we suggest that you run the configure and make processes as your normal UNIX user ID. The `make install` must be run as root. But if your normal ID has an environment setup for using the C compiler, it's simpler to do that than to set up root to have the correct environment.

Normally, we support the GCC compiler, but we have also attempted to support the Solaris workshop compilers and EGCS compilers. Please be aware that if you do not use GCC, you may experience a few problems.

Whichever compiler you do have, please insure that you can execute the compiler from the command line before running configure. If you do not have an environment setup to run the compiler first, configure will fail.

Before running `./configure`, please be sure that you do not have /usr/ucb on your path. This may cause the `./configure` to choose the wrong shutdown program. If `./configure` detects that /usr/usb is on your path, it will print a warning message. Please follow the advice to avoid shutdown problems.

Your normal UNIX user ID must own the source tree directories, and you must have the normal development tools in your path. This includes make, the compiler, the M4 preprocessor, the linker, and ar or ranlib. If the user you are logged in as can compile and link a C program from a source file, then you have all the required tools available.

You will want to install the executables in a directory that remains mounted during the shutdown. Solaris will unmount almost everything except the root directories. Since the ability to power the UPS off requires access to the executable programs, they need to be in a directory that will never be unmounted. And since they should also be in a directory that normal users cannot get into, /sbin is the default. However, please be aware that if you want to follow Sun's filesystem conventions you would use the following:

```
./configure \
   --prefix=/opt/apcupsd \
   --sbindir=/etc/opt/apcupsd/sbin \
   --sysconfdir=/etc/opt/apcupsd \
   --with-cgi-bin=/opt/apcupsd/cgi-bin
```

The way to setup the /sbin directory as the executables directory is to pass configure the **sbindir=/sbin** option. No other arguments should be required, and your setup and platform should be detected automatically by configure.

Once you have run configure, you will need to do a `make`. Once the make has completed with no errors, you must su to root to complete the install. After the su, you may not have a path to the make program anymore. In that case, you should do the `make install` step as:

```
/usr/ccs/bin/make install
```

Once the install completes, you must edit the /sbin/rc0 script as detailed

below, then exit from the su'ed shell.

In order to support unattended operation and shutdown during a power failure, it's important that the UPS remove power after the shutdown completes. This allows the unattended UPS to reboot the system when power returns by re-powering the system. Of course, you need autoboot enabled for your system to do this, but all Solaris systems have this by default. If you have disabled this on your system, please re-enable it.

To get the UPS to remove power from the system at the correct time during shutdown, i.e., after the disks have done their final sync, we need to modify a system script. This script is /sbin/rc0.

We do not have access to every version of Solaris, but we believe this file will be almost identical on every version. Please let us know if this is not true.

At the very end of the /sbin/rc0 script, you should find lines just like the following:

```
# unmount file systems. /usr, /var and /var/adm are not unmounted by umountall
# because they are mounted by rcS (for single user mode) rather than
# mountall.
# If this is changed, mountall, umountall and rcS should also change.
/sbin/umountall
/sbin/umount /var/adm >/dev/null 2>\&1
/sbin/umount /var >/dev/null 2>\&1
/sbin/umount /usr >/dev/null 2>\&1

echo 'The system is down.'
```

We need to insert the following lines just before the last 'echo':

```
#see if this is a powerfail situation
if [ -f /etc/apcupsd/powerfail ]; then
        echo
        echo "APCUPSD will power off the UPS"
        echo
        /etc/apcupsd/apccontrol killpower
        echo
        echo "Please ensure that the UPS has powered off before rebooting"
        echo "Otherwise, the UPS may cut the power during the reboot!!!"
        echo
fi
```

We have included these lines in a file called rc0.solaris in the distribu-

tions/sun subdirectory of the source tree. You can cut and paste them into the /sbin/rc0 file at the correct place, or yank and put them using vi or any other editor. Note that you must be root to edit this file.

You must be absolutely sure you have them in the right place. If your /sbin/rc0 file does not look like the lines shown above, do not modify the file. Instead, email a copy of the file to the maintainers, and we will attempt to figure out what you should do. If you mess up this file, the system will not shut down cleanly, and you could lose data. Don't take the chance.

This feature has only been tested with APC SmartUPS models. If you do not have a SmartUPS, you will be one of the first testers to try this feature. Please send email to let us know if it works with your UPS model, what model you have, and if possible, the event logs located in /etc/apcupsd. We'd be very interested in your results, and would be glad to work with you to get this feature working correctly with all the APC models. A detailed description of the screen output during the shutdown would be very helpful if you see problems.

You will then need to make the normal changes to the /etc/apcupsd/apcupsd.conf file. This file contains the configuration settings for the package. It is important that you set the values to match your UPS model and cable type, and the serial port that you have attached the UPS to. People have used both /dev/ttya and /dev/ttyb with no problems. You should be sure that logins are disabled on the port you are going to use, otherwise you will not be able to communicate with the UPS. If you are not sure that logins are disabled for the port, run the 'admintool' program as root, and disable the port. The 'admintool' program is a GUI administration program, and required that you are running CDE, OpenWindows, or another XWindows program such as KDE.

Solaris probes the serial ports during boot, and during this process, it toggles some handshaking lines used by dumb UPSes. As a result, particularly for simple signalling "dumb" UPSes it seems to kick it into a mode that makes the UPS think it's either in a calibration run, or some self-test mode. Since at this point we are really not communicating with the UPS, it's pretty hard to tell what happened. But it's easy to prevent this, and you should. Disconnect the UPS, and boot the system. When you get to a login prompt, log in as root. Type the following command:

```
eeprom com1-noprobe=true
```

or

```
eeprom com2-noprobe=true
```

depending on which com port your UPS is attached to. Then sync and shutdown the system normally, reattach the UPS, and reboot. This should solve the problem. However, we have some reports that recent versions of Solaris (7 & 8) appear to have removed this eeprom option and there seems to be no way to suppress the serial port probing during boot.

At this point, you should have a complete installation. The daemon will load automatically at the next boot. Watch for any error messages during boot, and check the event logs in /etc/apcupsd. If everything looks OK, you can try testing the package by removing power from the UPS. NOTE! if you have a voltage-signalling UPS, please run the first power tests with your computer plugged into the wall rather than into the UPS. This is because dumb serial-port UPSes have a tendency to power off if your configuration or cable are not correct.

As a user, your input is very helpful in solving problems with the package, and providing suggestions and future directions for the development of the package. We are striving to provide a useful package that works across all platforms, and welcome your feedback.

Best regards, and thanks for your interest and help, The Apcupsd Development Team.

2.7.11 Unknown System: During the `./configure`, if apcupsd does not find one of the systems for which it has specific installation programs, it will set the Operating System to **unknown** and will use the incomplete installation scripts that are in <src>/distributions/unknown/. You will be on your own, or you can ask the developers list (apcupsd-users at lists.sourceforge.net) for installation instructions. This directory also contains a hint file for *Linux From Scratch*, which could be helpful for other systems as well.

2.7.12 Windows Systems with CYGWIN Installed: If you wish to build from the source, and if you have CYGWIN version 1.5.5 and GCC 2.95.3-5 installed, it is possible to build the Win32 version of apcupsd. Please don't try any other versions of CYGWIN as there were known problems.

To date, the Win32 version has only been build on a Win98 SR2 and a WinXP system with the above CYGWIN environment and all the available CYGWIN tools loaded. In addition, the builds were done running under the

bash shell. As time permits, we will experiment with other environments, and if any of you do build it from source, please let us know. The current CYGWIN environment was loaded using the CYGWIN setup.exe program, downloading ALL the latest binaries and installing them.

We recommend that you run the `./configure` command with the following options:

```
./configure \
  --prefix=/apcupsd \
  --sbindir=/apcupsd/bin \
  --sysconfdir=/apcupsd/etc/apcupsd \
  --with-pid-dir=/apcupsd/etc/apcupsd \
  --mandir=/apcupsd \
  --with-cgi-bin=/apcupsd/etc/apcupsd/cgi \
  --enable-pthreads
```

After which, you can do a:

```
make
```

And to install apcupsd, do:

```
make install
```

Finally, you should follow the Win32 (see Installation on Windows) installation instruction, skipping the part that describes unZipping the binary release.

3 After Installation

3.1 Checking Your Configuration File

Once you have installed apcupsd, either from a binary package or by building from source, your next step should be to inspect your /etc/apcupsd/apcupsd.conf file to make sure it is valid.

You can read the complete reference on configuration directives (see Configuration Directive Reference), but if you are setting up a normal standalone configuration you should only need to check (and possibly fix) the first three items listed below.

Your UPSTYPE should be the UPS's protocol type: dumb, apcsmart, usb, net, snmp, or ether. Your UPSCABLE should be the type of cable you are using. You should have gotten both from the table of types (see type_table); usually they will both be the string "usb".

If you have a USB device, it is better not to specify a DEVICE directive by commenting it out. Apcupsd will automatically search for your device in the standard places. If you specify a DEVICE, it should be the name of the device (or device range) that apcupsd is to use to communicate with the UPS. If you're using a USB UPS under Linux, you may leave the device name field blank and apcupsd will search all the standard locations for the UPS. You may also explicitly specify the device location as either /dev/usb/hid/hiddev[0-15] (on non-Red-Hat systems) or /dev/usb/hiddev[0-15] (on Red Hat systems), but this is not recommended.

Note that you should enter "/dev/usb/hiddev[0-15]" literally as shown. The "[0-15]" expression tells apcupsd to search all hiddev devices until it finds a UPS. You can restrict the search to a subset of devices by using something like "[0-4]", but keep in mind this will limit apcupsd's ability to locate the UPS if the kernel relocates it to a different device node, which happens occasionally during short power failures. Again, it is highly recommended to leave the DEVICE directive blank and let apcupsd find your device automatically.

If the first time you execute apcupsd, you get a message to the effect that the Apcupsd USB driver is missing, it means that you most likely forgot to put *—enable-usb* on your ./configure command line. If you loaded apcupsd from an rpm file, you may have selected the wrong one — please ensure that the word *usb* appears in the rpm package name.

The next chapter (see Configuration Examples) of this manual provides you with the essential characteristics of each main type of configuration file. After those elements are correct, apcupsd should run, and then it is only a matter of customization of your setup.

3.2 Arranging for Reboot on Power-Up

The final consideration for a automatic reboot after a full power down is to ensure that your computer will automatically reboot when the power is restored.

This is not the normal behavior of most computers as shipped from the factory. Normally after the power is cut and restored, you must explicitly press a button for the power to actually be turned on. You can test your

computer by powering it down; shutting off the power (pull the plug); then plugging the cord back in. If your computer immediately starts up, good. There is nothing more to do.

If your computer does not start up, manually turn on the power (by pressing the power on button) and enter your computer's SETUP program (often by pressing DEL during the power up sequence; sometimes by pressing F10). You must then find and change the appropriate configuration parameter to permit instant power on.

Normally, this is located under the **BOOT** menu item, and will be called something such as **Restore on AC/Power Loss** or **Full-On**. The exact words will vary according to the ROM BIOS provider. Generally you will have three options: **Last State**, **Power On**, and **Power Off**. Although **Last State** should normally work, we recommend setting your computers to **Power On**. This means that whenever the power is applied they are on. The only way to shut them off is to pull the plug or to have a special program that powers them off (/sbin/poweroff on Linux systems).

If after making all the changes suggested above, you cannot get your computer to automatically reboot, you might examine your halt script (/etc/rc.d/init.d/halt in the case of Red Hat Linux) and see if the final line that performs the halt or reboot contains the `-p` option for powering down the computer. It should not with the logic used by apcupsd, but if it does, the `-p` option could cause your computer to power off while the UPS is still suppling power (i.e. before the UPS kills the power). Depending on the setting of your BIOS, it may prevent your computer from restarting when the power returns. As already mentioned, this should not apply, but in case of problems it is worth a try.

3.3 Making sure apcupsd Is Running

The simplest way to invoke apcupsd is from the command line by entering:

```
/sbin/apcupsd
```

To do so, you must be root. However, normally, you will want apcupsd started automatically when your system boots. On some systems with installation support (e.g. SUSE and Red Hat), the installation procedure will create a script file that you will be automatically invoked when your system reboots. On other systems, you will have to invoke apcupsd from your rc.local script.

On Red Hat systems, this script file that automatically invokes apcupsd on system start and stops is: /etc/rc.d/init.d/apcupsd

To start apcupsd manually (as you will probably do immediately following the installation), enter the following:

```
/etc/rc.d/init.d/apcupsd start
```

To understand how this file is automatically invoked at system startup and shutdown, see the man pages for chkconfig(8).

On SUSE systems, the script file that automatically invokes apcupsd on system start and stops is /etc/rc.d/apcupsd

To start apcupsd manually (as you will probably do immediately following the installation), enter the following:

```
/etc/rc.d/apcupsd start
```

Normally, when properly installed, apcupsd will be started and stopped automatically by your system. Unfortunately, the details are different for each system. Below, we give the commands for selected systems. Alternatively, there are simple stopapcupsd and startapcupsd scripts in the examples directory, or you can modify one of the scripts in the distributions directory to meet your needs.

To stop apcupsd you can do the following:

On Red Hat systems:

```
/etc/rc.d/init.d/apcupsd stop
```

On SUSE systems:

```
/etc/rc.d/apcupsd stop
```

Please see the Testing Apcupsd (see Testing Apcupsd) chapter for more details on insuring that apcupsd is running properly.

4 Configuration Examples

4.1 A Simple USB Configuration

If you have a USB UPS, and you have apcupsd version 3.10.7 or higher, the essential elements of your apcupsd.conf file should look like the following:

```
## apcupsd.conf v1.1 ##
UPSCABLE usb
UPSTYPE usb
DEVICE
LOCKFILE /var/lock
UPSCLASS standalone
UPSMODE disable
```

Notice that we have not specified a device. In doing so, apcupsd will try all the well known USB ports. We strongly recommend you use this (empty device address) form unless you have a good reason to do otherwise.

An alternate way of specifying the device is to specify a range of device addressess as follows:

```
DEVICE /dev/usb/hid/hiddev[0-15]
```

If you have more than one device, you may need to specify each device individually using absolute device paths. This is not, however, recommended.

```
DEVICE /dev/usb/hiddev0
```

Please use the explicit specifications of a device only if your know exactly what you are doing. In general, it is much easier to let apcupsd find the device itself.

If you use the range specification, you should enter **/dev/usb/hiddev[0-15]** literally as shown. The "[0-15]" expression tells apcupsd to search all hiddev devices until it finds a UPS. You can restrict the search to a subset of devices by using something like "[0-4]", but keep in mind this will limit apcupsd's ability to locate the UPS if the kernel relocates it to a different device node.

On Debian systems, the **hiddev** devices are not automatically defined. As a consequence, you will need to run the **make-hiddev** script in the **examples** directory of the source.

4.2 A Simple Configuration for a SmartUPS

If you have a Smart UPS using the cable supplied by APC, or you build a CUSTOM SMART cable outlined in the cables chapter, a very simple configuration file would look like the following:

```
## apcupsd.conf v1.1 ##
UPSCABLE smart
UPSTYPE smartups
DEVICE /dev/ttyS0
LOCKFILE /var/lock
UPSCLASS standalone
UPSMODE disable
```

Normally you would have many more configuration directives to completely customize your installation, but this example shows you the minimum required.

4.3 A Simple Configuration for a Simple Signaling or Dumb

If you have a simple signaling or dumb UPS such as a BackUPS, you will need to know exactly what cable you have and specify it on the UPSCABLE directive. Please see the list of UPSes versus cables in the beginning of this document for more information. The cable number is normally stamped in the plastic at one end of the cable. If you specify the wrong cable, it is very likely that at the first power failure, your computer will be immediately shutdown. This is an unfortunate consequence of the dumb signaling mode. To avoid this, first replace **/etc/apcupsd/apccontrol** with **safe.apccontrol** found in the examples directory, then test until everything works correctly. Once you have the correct cable, be sure to remember to reinstall the correct apccontrol file and test that your computer is correctly shutdown during a power failure.

```
## apcupsd.conf v1.1 ##
UPSCABLE (number of cable you have)
UPSTYPE dumb
DEVICE /dev/ttyS0
LOCKFILE /var/lock
UPSCLASS standalone
UPSMODE disable
```

If your cable does not have low battery detection, as is the case with some older models, you will also need to define **TIMEOUT nnn** where you set

nn to be the number of seconds on a power failure after which a shutdown is effected.

Normally you would have many more configuration directives to completely customize your installation, but this example shows you the minimum required.

4.4 A Simple Master Configuration

You have a Smart UPS using the cable supplied by APC and you want it to act as a master for another computer, which is powered by the same UPS. A very simple configuration file would look like the following:

```
## apcupsd.conf v1.1 ##
UPSCABLE smart
UPSTYPE smartups
DEVICE /dev/ttyS0
LOCKFILE /var/lock
UPSCLASS netmaster
UPSMODE net
NETTIME 10
NETPORT 6666
SLAVE slave1.mynetwork.com
SLAVE slave2.mynetwork.com
```

Note, the main difference from the stand alone configuration is that you have specified **UPSCLASS netmaster** and **UPSMODE net**. In addition, you have specified one or more slave machines. In this mode of networking, (as opposed to using the net driver as described several sections below), your master knows the presence of all the slaves. They carry on a very explicit communication, and the slaves are explicitly notified by the master of any important changes such as a shutdown.

There is a simpler form of contolling slaves using the net driver with an apcupsd NIS server. The simpler form is much easier to configure. See: see A Sample NIS Slave Configuration Using the Net Driver below for details.

4.5 A Simple Slave Configuration

You have a Smart UPS using the cable supplied by APC that is connected to the master machine configured above, and the master machine is running as a netmaster and has the address of your slave machine. This slave machine

has no serial port connection to the UPS, but is powered by the same UPS as the master. A very simple configuration file would look like the following:

```
## apcupsd.conf v1.1 ##
UPSCABLE ether
UPSTYPE smartups
LOCKFILE /var/lock
UPSCLASS netslave
UPSMODE net
NETPORT 6666
MASTER master.mynetwork.com
```

The main difference from the master configuration is that you have specified **UPSCABLE ether** and **UPSCLASS netslave**. In addition, you have specified a single controlling master.

Please note, there are reports that you **must** use **UPSTYPE smartups** on the slave even if the master is using **UPSTYPE dumb**. This is apparently some bug in the new dumb driver.

In this configuration, the shutdown will be initiated by the master. It is also possible to specify BATTERYLEVEL, MINUTES, and TIMEOUT configuration directives in the Slave machine that will cause the slave to shutdown before the master. This can often be useful if the slave is less important than the master and you wish to reduce battery power consumption so that the master can remain up longer during a power outage.

4.6 Variation on the Master/Slave Configuration

It is also possible to have a Master/Slave configuration where the Slave is powered by a different UPS (or any other power source), but is nevertheless controlled (i.e. shutdown) by the master. The setup would be identical to the Master/Slave configuration files shown above. The only difference is where the slave actually receives its power. In effect, apcupsd does not know or care where the power really comes from.

4.7 A Sample NIS Slave Configuration Using the Net Driver

As opposed to the old master/slave mode demonstrated above, you can turn any computer into an NIS slave by configuring with the NIS network driver turned on `--enable-net`. The difference is that the NIS server has no explicit knowledge of the slaves. The NIS server makes its information

available via the net (NIS), and the NIS slaves read it. When the NIS server is going to shutdown, it makes the information available to any NIS slave that polls it, but the NIS server does not explicitly call each NIS slave as is the case in the Master/Slave networking described several sections above.

Running in this configuration, you can use any computer with apcupsd running the Network Information Server (NIS) as the server. The NIS slave simply uses the NIS information to decide when to shutdown. This is a much simpler mode than the older master/slave code mentioned above.

The main apcupsd (NIS server) is connected to the UPS and has NIS turned on, but the configuration is a simple standalone as in the section **A Simple Configuration for a SmartUPS**. It doesn't matter how the UPS is connected to the computer (serial, USB, ...).

For the NIS slave computer, you will have a configuration that looks something like what follows. What is important is that you get the information from an **ether** cable over the network and you must specify the address of a "NIS server" that is running NIS (not the Master/Slave networking described above). The NIS slave apcupsd will then poll the NIS server at the NETTIME interval you specify to obtain the status.

Here are a few words from Adam Kropelin concerning the difference between the Master/Slave networking and the NIS-based networking:

Think of the difference as push (Master/Slave) vs. pull (NIS-based). In the case of M/S, the master makes all the shutdown decisions and notifies the slaves when they are to shut down or when some other interesting event happens. The slaves just do whatever the master says, whenever the master says to. On the other hand, with the NIS-based network config you basically "publish" the UPS status from one server and then your clients view that status and make their own decisions.

Personally, I like the NIS-based approach because the master knows nothing about the slaves, thus there are fewer configuration files to keep in sync. I also like the flexibility of allowing each slave to make its own decision on when to shut down; some of my old clunker servers take quite a long while to shut down. There are problems reported occasionally with the M/S approach, where slaves sometimes lose contact with the master or vice-versa. I know improvements have been made in that code, but I still like the simplicity of using NIS.

Another thing to think about is how you feel about running a network service like NIS on your firewall. My network is set up almost identically to yours and I chose to run the apcupsd "master" on a server in the DMZ and have

the firewall just be a client of it. That way I don't have to run NIS on the firewall apcupsd instance.

```
## apcupsd.conf v1.1 ##
UPSCABLE ether
UPSTYPE net
LOCKFILE /var/lock
DEVICE server-network-address:3551
UPSCLASS standalone
UPSMODE disable
NETTIME 10
```

where on the **DEVICE** directive you replace the `server-network-address` with the fully qualified domain name or IP address of a machine running apcupsd with NIS enabled (and normally, but not required, connected to a UPS). The :3551 that follows the NIS server address is the port to use. The default is 3551, but older versions of apcupsd used port 7000.

Please do not confuse this NIS server/slave mode with the old master/slave network configuration that is described above. This is a master/slave setup, but much simpler (the NIS server does not know about the slaves), and any NIS server, even a slave, can act as a server to a slave that listens to it.

The **NETTIME** directive defines the time interval that the slave uses to poll the NIS server. If you set this too large, your slave may not see the change in state of the NIS server before the server has shutdown. Normally, you have at least 30 seconds of grace time between the time the NIS server decides to shutdown and the time it no longer responds. Your slave must poll during this interval.

This mode works principally by reading the STATFLAG record that is sent by the NIS (present in the output of apcaccess). The low 16 bits are the standard APC status flag, and the upper 16 bits represent the internal state of apcupsd, so the slave can see when the power fails and know when to shutdown.

As with the Master/Slave configuration, any slave run using the Net driver will shutdown when its own timers expire or when the NIS server shuts down, whichever occurs first. This means that if you want the slave to shutdown before the server, you need only set BATTERYLEVEL, or any of the other values on the slave for a faster shutdown than the values defined on the NIS server.

5 Testing Apcupsd

The following testing procedures apply for the most part to apcsmart UPSes, whether USB or serial. If you have a dumb voltage-signalling UPS, your testing procedures will be somewhat different, and you should see the section on Testing Serial UPSes (see Testing Serial-Line UPSes).

5.1 Process-Status Test

After you start apcupsd, execute the following command:

```
ps fax
```

or the equivalent for your system. If you are running on Linux and using the fork()ing version of apcupsd, you should something similar to the following output.

```
4492 ?        S      0:00 apcmain       -f /etc/apcupsd/apcupsd.conf
4496 ?        S      0:00  \_ apcser        -f /etc/apcupsd/apcupsd.conf
4497 ?        S      0:00  \_ apcnis        -f /etc/apcupsd/apcupsd.conf
```

This indicates that apcupsd is up and running and has started the two (default) child processes. If you are running with the pthreaded version, now the default, and 2.4.x kernels, you will still see the three processes (see below). However, under 2.6.x kernels, the threads do not have independent process ids so everything will be compressed into a single **ps** line.

apcmain is the main program that waits until it receives a termination signal (SIGTERM) or one of the child processes dies.

apcser is the process that manages the serial port and takes any actions (generates events) that are necessary as a result of a change of state of the UPS.

apcnis is the Network information server process that provides EVENTS and STATUS information over the network. This information is used by the CGI programs.

If you are running on a non-Linux system, or using pthreads on a Linux system (recommended), your output will probably not show the names of the processes and will appear more like the following:

```
632 ?        S      0:00 /sbin/apcupsd -f /etc/apcupsd/apcupsd.conf
841 ?        S      0:00  \_ /sbin/apcupsd -f /etc/apcupsd/apcupsd.conf
842 ?        S      0:00      \_ /sbin/apcupsd -f /etc/apcupsd/apcupsd.conf
```

If you see only one instance of apcupsd running, don't worry about it as this is normal on most non-Linux systems, and on Linux 2.6.x kernels.

If you do not find that apcupsd is in the above list, the most likely problem is a configuration file glitch. If no messages were printed, you should check your system log (normally /var/log/messages where you will find one or messages indicating the nature of the problem.

5.2 Logging Test

Once you have established that the proper processes are running, do a tail of the system log file, normally /var/log/messages:

```
tail /var/log/messages
```

You should see output that looks similar to the following:

```
Dec 5 17:01:05 matou apcupsd[5917]: apcupsd 3.7.2
startup succeeded
```

And if you have configured the network information server, you should also see:

```
Dec 5 17:01:05 polymatou apcupsd[5975]: apcserver
startup succeeded
```

These messages should also appear in the temporary file (/etc/apcupsd/apcupsd.events) if you are using the default configuration file. If you have installed the RPM, they will probably be in /var/log/apcupsd.events.

5.3 apcaccess Test

This test consists of running `apcaccess` to see if apcupsd is properly updating its internal variables. Please note that if you are running a pthreaded version of apcupsd, which you should be since the non-pthreaded version is no longer supported, (installed from rpm or `--enable-pthreads` on the `./configure` line), you must enable the apcupsd Network Information Server in your configuration file for **apcaccess** to work. This is done by setting:

```
NETSERVER on
NISPORT 3551
```

in your **apcupsd.conf** file.

To run the apcaccess test, use the following command:

```
apcaccess status
```

Depending on the type of UPS you have, you will get slightly different output, but an example For a Smart-UPS is as follows:

```
APC      : 001,048,1088
DATE     : Fri Dec 03 16:49:24 EST 1999
HOSTNAME : daughter
RELEASE  : 3.7.2
CABLE    : APC Cable 940-0024C
MODEL    : APC Smart-UPS 600
UPSMODE  : Stand Alone
UPSNAME  : SU600
LINEV    : 122.1 Volts
MAXLINEV : 123.3 Volts
MINLINEV : 122.1 Volts
LINEFREQ : 60.0 Hz
OUTPUTV  : 122.1 Volts
LOADPCT  :  32.7 Percent Load Capacity
BATTV    : 26.6 Volts
BCHARGE  : 095.0 Percent
MBATTCHG : 15 Percent
TIMELEFT :  19.0 Minutes
MINTIMEL : 3 Minutes
SENSE    : Medium
DWAKE    : 000 Seconds
```

```
DSHUTD   : 020 Seconds
LOTRANS  : 106.0 Volts
HITRANS  : 129.0 Volts
RETPCT   : 010.0 Percent
STATFLAG : 0x08 Status Flag
STATUS   : ONLINE
ITEMP    : 34.6 C Internal
ALARMDEL : Low Battery
LASTXFER : Unacceptable Utility Voltage Change
SELFTEST : NO
STESTI   : 336
DLOWBATT : 05 Minutes
DIPSW    : 0x00 Dip Switch
REG1     : N/A
REG2     : N/A
REG3     : 0x00 Register 3
MANDATE  : 03/30/95
SERIALNO : 13035861
BATTDATE : 05/05/98
NOMOUTV  : 115.0
NOMBATTV :  24.0
HUMIDITY : N/A
AMBTEMP  : N/A
EXTBATTS : N/A
BADBATTS : N/A
FIRMWARE : N/A
APCMODEL : 6TD
END APC  : Fri Dec 03 16:49:25 EST 1999
```

For a simple signaling or dumb UPS such as BackUPS, your output will be very minimal as follows:

```
APC      : 001,012,0319
DATE     : Mon Feb 18 09:11:50 CST 2002
RELEASE  : 3.8.5
UPSNAME  : UPS_IDEN
CABLE    : APC Cable 940-0128A
MODEL    : BackUPS
UPSMODE  : Stand Alone
STARTTIME: Mon Feb 18 09:11:45 CST 2002
LINEFAIL : OK
BATTSTAT : OK
STATFLAG : 0x008 Status Flag
END APC  : Mon Feb 18 09:15:01 CST 2002
```

If you see the above output, it is a good sign that apcupsd is working. Assuming that the output looks reasonable, check the following variables:

A very disturbing tendance is for some of the newer (Mar 2004) RS and ES UPSes to have no Voltage information. This is annoying bug not serious. On

the other hand, some of those UPSes now have no battery charge information (BCHARGE). If BCHARGE is zero in your listing and you are running a Smart or a USB UPS, then you will have to set the BATTERYLEVEL directive in your apcupsd.conf file to -1.

LINEV This is the line voltage and it should be a value that is appropriate for your equipment. In the USA, it is typically about 120 Volts while in Europe, it is about 220 Volts.

BATTV Unless you have additional battery packs, this should be near 24 Volts plus or minus 5 Volts.

STATUS This is the status of the UPS and it should normally be **ONLINE**.

If you see a message to the effect of:

```
attach_shmarea: shared memory version mismatch (or UPS not yet ready to report)
```

or if all the displayed values are zero, you have not waited long enough. Wait a bit longer and then re-execute the `apcaccess status` command.

If you see a message to the effect of:

```
APCACCESS FATAL ERROR in apcaccess.c at line 336
tcp_open: cannot connect to server localhost on port 3551.
```

It means that you have probably not enabled the Network Information Server in your configuration file for **apcaccess** to work. This is done by setting:

```
NETSERVER on
NISPORT 3551
```

in your **apcupsd.conf** file.

5.4 Communications Test

At this point, you should ensure that apcupsd is handling the connection to the UPS correctly. This test assumes you have a UPS that speaks apcsmart protocol, over either USB or a serial port. If you have an old-style voltage-signaling UPS, please skip to the next section (Simulated Power Fail Test).

When apcupsd detects a problem, it generates an EVENT, which consists of sending a message to the system log then invoking the **apccontrol** script (normally in /etc/acpupsd/apccontrol) to handle the event.

In order to create an event, remove the serial port plug from the back of your computer or from the back of the UPS. Within 6 seconds, apcupsd should detect the lack of serial port communications and broadcast a `wall` message indicating that the serial port communications was lost:

Warning communications lost with UPS lost.

At the same time, it sends the same message to the system log and to the temporary EVENTS file (/etc/apcupsd/apcupsd.events).

Plug the serial port plug back into your computer, and within about 12 seconds, apcupsd should reestablish communications and broadcast and log the following message:

Communications with UPS restored.

If these messages are logged but not broadcast, either you have your mesg permission set to *no* (see **man wall** or **man mesg**), or there is a problem with apccontrol. If you are running a window manager such as GNOME and don't have a console window open, you may not receive the `wall` messages. However, you should find them in your system log file (normally /var/log/messages and in the temporary EVENTS file, /etc/apcupsd/apcupsd.events. For example, to observe these events in the temporary EVENTS file, you might do a

```
tail -f /etc/apcupsd/apcupsd.events
```

Note, if you have installed from the RPM, the proper events file may be /var/log/apcupsd.events. You can find the actual filename by checking your apcupsd.conf file.

before running the test.

If you do not observe these messages, you should correct this problem before

proceeding with additional tests.

5.5 Simulated Power Fail Test

At this point, you should verify that in the event of a power fail apcupsd properly calls apccontrol. This test is appropriate for all models of UPSes (smart or dumb).

To avoid the possibility that apcupsd might shut down your system, locate where apccontrol resides on your system (normally, /etc/apcupsd/apccontrol. Move this script to another location e.g. apccontrol.save and replace it with the script found in examples/safe.apccontrol. When that is done, ensure that your UPS battery is fully charged and that you have at least 5 minutes of remaining runtime on the batteries. This can be done by examining the values of the **BATTCHG** and **TIMELEFT** variables in the printout of `apcaccess status`.

Athough this should not be necessary, as an extra precaution, you can shutdown your machine, remove the plug from the UPS you are testing, and plug your machine into another UPS or directly into the wall. Doing so, will ensure that the UPS doesn't cut the power to your machine at a bad time. Remember at the end of the testing to plug your machine back into the UPS.

You can also minimize the risk from an unexpected shutdown by using a journaling filesystem such as Linux's EXT3. All modern disk drives park themselves safely when they power down, rather than ploughing up oxide on your disk's recording surface. Thus, unexpected power less has to hit very narrow timing windows in order to trash an EXT3 transaction.

To begin the test, pull the power plug from the UPS. The first time that you do this, psychologically it won't be easy, but after you have pulled the plug a few times, you may even come to enjoy it. If all goes well, apcupsd should detect the power failure and print several warning messages. The first should appear after 5 to 6 seconds and read:

```
Warning power loss detected.
```

Then generally 6 seconds later, apcupsd is sure that it isn't a transient effect, so it sends:

```
Power failure. Running on UPS batteries.
```

After a few more seconds (total around 15 seconds), plug the power cord back in and ensure that apcupsd is aware that the power has returned. It should print:

```
Power has returned...
```

If you do not observe the above messages, please correct the situation before proceeding. The most likely cause of problems are:

- apcupsd doesn't recognize the power failure because the configuration directives are not correct. E.g. wrong cable.
- The file /etc/apcupsd/apccontrol doesn't exist or is not marked as executable.

At this point, we recommend that you do a simulated power down of your system. If you are adventuresome or have been through this before, skip to the next section in this manual and do the real power fail shutdown. If you continue with the simulated power down and if all goes well, apcupsd will go through all the motions without actually shutting down the system. Continue using the safe apccontrol that you installed. Edit the configuration file apcupsd and change the value of **TIMEOUT** from 0 to something like 30. Doing so will cause apcupsd to attempt to shutdown the system 30 seconds after it detects a power failure. Once this change has been made, you must stop and restart apcupsd for the new configuration value to take effect.

Once again, pull the power plug, and if all goes as expected, apcupsd should attempt to shutdown the system about 30 seconds after it detects the power failure. All the messages should be displayed by `wall` or by the `tail -f` command. The precise message is determined by what is printed in /etc/apcupsd/apccontrol for the **doshutdown** event. Though it varies from system to system, it will generally be something like:

```
Beginning Shutdown Sequence
```

When apcupsd this message prints, reconnect the power. apcupsd should detect that the power has been restored and attempt to cancel the shutdown.

IMPORTANT after this test, please replace the changed apccontrol and apcupsd.conf with the original files.

5.6 System Shutdown Test

This is an intermediate test that you can do, for all UPS models before doing the Full Power Down Test. First modify the /etc/apcupsd/apccontrol file so that in the **killpower)** case, the line that re-executes apcupsd with the `--killpower` option is commented out. The original line probably looks something like:

```
${APCUPSD} --killpower
```

when it is commented out, it looks like:

```
#${APCUPSD}--killpower
```

Now when you pull the power plug, and either the timer expires or the batteries are exhausted (see the next section for more details), the system should be fully shutdown.

After performing this test, please be sure to restore /etc/apcupsd/apccontrol to its previous state.

5.7 Full Power Down Test

To complete the testing, you should do a power fail shutdown of your system. This test is applicable to all UPS models. Please do a backup of your system or take other precautions before attempting this to avoid the possibility of lost data due to a problem (I have been through this at least 10 times and never once had problems, but we all know that someday something will go wrong).

Before proceeding, please ensure that your halt script or the equivalent has been properly updated by the install process to contain the logic to call `apcupsd --killpower` when it detects a power failure situation (the presence of a /etc/powerfail file). See the Building and Installing apcupsd of this manual, or the README files for additional details about the halt modifications necessary.

When you are ready to do the test, either simply pull the plug and wait for the batteries to become exhausted, or set the **TIMEOUT** configuration directive to something like 60 so that the system will shutdown before the batteries are exhausted. We recommend doing the full shutdown without using **TIMEOUT** to correctly simulate a real power failure, but the choice is yours (I did it once here, but now use TIMEOUT 30).

If all goes well, your system should be shutdown before the batteries are completely exhausted and the UPS should be powered off by apcupsd. Please be aware that if you do the full power down, you must ensure that your UPS is totally powered off. Otherwise, it may have been given the command to power off, but due to a long grace period it is still waiting. If you were to reboot your computer during the grace period, the UPS could then suddenly turn off the power (this happened to me). To avoid this problem, always wait for your UPS to power itself off, or power if off manually before restarting your computer. On my system, the UPS is configured as at the factory to have a 180 second grace period before shutting off the power. During this type of testing, 180 seconds **seems** like an eternity, so please take care to either wait or manually power off your UPS. To determine what grace period is programmed into your UPS EEPROM, run `apcaccess eprom` and look at the "Shutdown grace delay".

5.8 Shutdown Sequence

If you experienced so problems with the above testing procedures, or if you are porting apcupsd to another system, or you are simply curious, you may want to know exactly what is going on during the shutdown process. If so, please see the Shutdown Sequence (see Shutdown Sequence <1>) section of this manual.

5.9 apctest

apctest is a program that allows you to talk directly to your UPS and run certain low-level tests, display all know values from the UPS's EEPROM, perform a battery runtime calibration, program the EEPROM (serial connection only), and enter in TTY mode with the UPS. Here we describe how to use it for a USB or apcsmart UPS; see Using apctest on Serial-Line UPSses for a description of how to use it with a voltage-signalling UPS.

Shutdown apcupsd if it is running. Make sure your **/etc/apcupsd/apcupsd.conf** file has **UPSTYPE smart** and **UPSCABLE** has one of the smart cables that are supported.

Normally apctest will have been built but not installed, so you must execute it from the <apcupsd-source>/src directory. You can explicitly build it on Unix with:

```
cd <apcupsd-source-directory>
make apctest
./apctest
```

or on Windows systems with:

```
make apctestwin32
./apctest
```

It will read your installed apcupsd.conf configuration (so it knows where to find the UPS) and then it will present you with the following output:

```
2003-07-07 11:19:21 apctest 3.10.6 (07 July 2003) redhat
Checking configuration ...
Attached to driver: apcsmart
sharenet.type = DISABLE
cable.type = CUSTOM_SMART

You are using a SMART cable type, so I'm entering SMART test mode
mode.type = SMART
Setting up serial port ...
Creating serial port lock file ...
Hello, this is the apcupsd Cable Test program.
This part of apctest is for testing Smart UPSes.
Please select the function you want to perform.

1) Query the UPS for all known values
2) Perform a Battery Runtime Calibration
3) Abort Battery Calibration
4) Monitor Battery Calibration progress
5) Program EEPROM
6) Enter TTY mode communicating with UPS
7) Quit

Select function number: 1
```

Item 1 will probe the UPS for all values known to apcupsd and present them in rather raw format. This output can be useful for providing technical support if you are having problems with your UPS.

Item 2 will perform a Battery Runtime Calibration. This test will only be performed if your battery is 100% charged. Running the test will cause the batteries to be discharged to approximately 30% of capacity. The exact number depends on the UPS model. In any case, apctest will abort the test if it detects that the battery charge is 20% or less.

The advantage of doing this test is that the UPS will be able to recalibrate the remaining runtime counter that it maintains in its firmware. As your batteries age, they tend to hold less of a charge, so the runtime calibration may not be accurate after several years.

We recommend that perform a Battery Calibration about once a year. You should not perform this calibration too often since discharging the batteries tends to shorten their lifespan.

Item 3 can be used to abort a Battery Calibration in progress, if you some how became disconnected.

Item 4 can be used to restart the monitoring of a Battery Calibration if you should some how become disconnected during the test.

Item 5 is used to program the EEPROM. Please see the Configuration Directives Used to Set the UPS EPROM chapter of this manual for the details.

Item 6 will initiate a direct communication between your terminal and the UPS at which point, you can enter raw UPS commands. Please be aware that you should be careful what commands you enter because you can cause your UPS to suddenly shutdown, or you can modify the EEPROM in a way to disable your UPS. The details of the raw Smart mode UPS commands can be found in the UPS Bible (see APC smart protocol) chapter of this manual.

Item 7 will terminate apctest.

6 Troubleshooting Your Installation

6.1 Known Problems with USB UPSes

6.1.1 Some Cheaper Models Do Not Have Battery Charge: Unfortunately, some cheaper USB models do not seem to report BCHARGE in the apcaccess output listing, which means with a standard conf file, your system will be immediately shutdown. To correct this, set the BATTERYLEVEL directive in your apcupsd.conf file to -1.

Some of these cheaper USB UPSes also do not report the Voltage. This is annoying but does not cause the unit to malfunction.

6.1.2 Reconnection does not clean up the lockfile: If either you disconnect the UPS or it disconnects because of some electrical problem, it will most certainly reconnect with a different device number. Apcupsd will detect this and reconnect properly. However, apcupsd does not release the old device (USB port) lock file and create a new one. This is not too serious.

6.1.3 Power Off (killpower) of UPS Does Not Work: Currently (as of 3.10.6) the code to power off the UPS works only if you have a Linux kernel version 2.4.22 or greater, or you have applied the patches in the examples directory to your kernel.

6.1.4 apcupsd Cannot Reconnect After a Reboot: If apcupsd does not connect to the USB port when you reboot, it is probably the appropriate kernel modules are not getting loaded correctly.

You can check this by bringing up your system, fiddling around until you get apcupsd to work with the UPS, then doing `cat /proc/modules` andnd save the output some place. Then reboot your computer and before you do anything else, do the `cat /proc/modules` again. Most likely you will find some of the usb modules are missing in the second listing.

There are two solutions:

- Ensure that you have the hotplug program loaded. It should fix the problem. This is a bit of magic, so we are not exactly sure how it works. The rpm I (Kern) have loaded is: hotplug-2001_02_14-15

  You might want to read the man page on hotplug, and it might be necessary to `cp /etc/hotplug/usb.rc /etc/init.d/hotplug` to get it fully working.

- You can explicitly force the appropriate usb modules to be loaded by adding:

  ```
  /sbin/modprobe <missing-module-name>
  ```

  in the /etc/rc.d/init.d/apcupsd script just after the **start)** case (at about line 17). This will force the modules to be loaded before apcupsd is invoked.

7 Monitoring and Tuning your UPS

After you have verified that your UPS is working correctly, you will probably want to query the state of its health occasionally. The tools apcupsd gives you to do this include one command-line utility (apcaccess) and a GUI you can use through a Web browser. You can also use apctest to tune some parameters of the UPS itself.

7.1 apcaccess

apcaccess is a program (normally found in /sbin/apcaccess) that permits you to print out the complete status of your UPS. Although there are a number of command line arguments (**eprom, reconfig, status, slave, shutdown**), all except **eprom** and **status** are under development and hence do not work reliably.

If you have built apcupsd with pthreads enabled (default), apcaccess will use the Network Information Server to obtain the necessary information for the **status** and **eeprom** commands. This is because in the pthreaded version, there is no IPC shared memory. In this case (pthreads enabled), you can specify a second optional argument to apcaccess in the form of host:port, where the :port is optional. The default is **localhost:3551**. Please note that in versions prior to 3.10.6, the default NIS port was 7000, so if you are mixing versions, you will need to take a lot of care to ensure that all components are using the same port.

To enable the apcupsd Network Information Server, which is normally the default, you set:

```
NETSERVER on
NISPORT 3551
```

in your **apcupsd.conf** file.

7.1.1 apcaccess status: As mentioned above, the full form of the command is:

```
apcaccess status localhost:3551
```

where only apcaccess status should normally be needed. localhost may be replaced by any machine name, fully qualified domain name, or IP address, which means that apcaccess can access any UPS on the network running the Network Information Server.

The **status** command line option of apcaccess will produce a full printout of all the **STATUS** variables used by apcupsd. This can be very helpful for checking the condition of your UPS and to know whether or not apcupsd is properly connected to it. For a complete description of the variables and their meanings, please read the Status Format (see apcupsd Status Logging) section of the Technical Reference.

Please note that if you invoke apcaccess within the first 30 seconds of launching apcupsd, you will likely get an error message such as:

```
APCACCESS FATAL ERROR in apcipc.c at line 325
attach_shmarea: shared memory version mismatch
```

This is because apcupsd is still in the process of initializing the shared memory segment used to communicate between the two processes. There is also a small window of time after which the memory segment is properly initialized but before the UPS has been completely polled. If you invoke apcaccess during this period, you will get the STATUS output, but with many of the values zero. The solution is to wait at least 30 seconds after starting apcupsd before launching apcaccess.

To invoke apcaccess, enter:

```
apcaccess status
```

For a SmartUPS 1000 apcaccess will emit the following output:

```
DATE     : Fri Dec 03 12:34:26 CET 1999
HOSTNAME : matou
RELEASE  : 3.7.0-beta-1
CABLE    : Custom Cable Smart
MODEL    : SMART-UPS 1000
UPSMODE  : Stand Alone
UPSNAME  : UPS_IDEN
LINEV    : 232.7 Volts
MAXLINEV : 236.6 Volts
MINLINEV : 231.4 Volts
```

```
LINEFREQ : 50.0 Hz
OUTPUTV  : 232.7 Volts
LOADPCT  :  11.4 Percent Load Capacity
BATTV    : 27.7 Volts
BCHARGE  : 100.0 Percent
MBATTCHG : 5 Percent
TIMELEFT : 112.0 Minutes
MINTIMEL : 3 Minutes
SENSE    : Low
DWAKE    : 060 Seconds
DSHUTD   : 180 Seconds
LOTRANS  : 204.0 Volts
HITRANS  : 253.0 Volts
RETPCT   : 050.0 Percent
STATFLAG : 0x08 Status Flag
STATUS   : ONLINE
ITEMP    : 29.2 C Internal
ALARMDEL : Low Battery
LASTXFER : U command or Self Test
SELFTEST : NO
STESTI   : 336
DLOWBATT : 02 Minutes
DIPSW    : 0x00 Dip Switch
REG1     : 0x00 Register 1
REG2     : 0x00 Register 2
REG3     : 0x00 Register 3
MANDATE  : 01/05/99
SERIALNO : GS9902009459
BATTDATE : 01/05/99
NOMOUTV  : 230.0
NOMBATTV :  24.0
HUMIDITY : N/A
AMBTEMP  : N/A
EXTBATTS : 0
BADBATTS : N/A
FIRMWARE : 60.11.I
APCMODEL : IWI
END APC  : Fri Dec 03 12:34:33 CET 1999
```

For the various smaller, cheaper APC USB UPSes, such as the CS, ES, ..., you will get much of the information that is presented above, but not all of it. For example, you will not get MAXLINEV, MINLINEV, LINEFREQ, ... and in particular, the LOADPCT will be zero when you are running on mains. LOADPCT will display when the UPS is on batteries. You must remember that the non-SmartUPSes are much simpler (and less expensive) and therefore produce less information.

7.1.2 apcaccess eprom: The **eprom** command line option for apcaccess allows you to examine the current values of your UPS' EPROM as well as to know the permitted values that can be set in the EPROM. For information

about changing these values, see the section on tuning EEPROM parameters (see Configuring Your EEPROM).

A typical output from `apcaccess eprom` is:

```
Valid EPROM values for the SMART-UPS 1000

                       Config        Current  Permitted
Description              Directive     Value    Values
================================================================
Upper transfer voltage   HITRANSFER    253      253 264 271 280
Lower transfer voltage   LOTRANSFER    208      196 188 208 204
Return threshold         RETURNCHARGE  15       00 15 50 90
Output voltage on batts  OUTPUTVOLTS   230      230 240 220 225
Sensitivity              SENSITIVITY   H        H M L L
Low battery warning      LOWBATT       2        02 05 07 10
Shutdown grace delay     SLEEP         180      020 180 300 600
Alarm delay              BEEPSTATE     T        0 T L N
Wakeup delay             WAKEUP        60       000 060 180 300
Self test interval       SELFTEST      336      336 168 ON  OFF
```

7.2 Apcupsd Notification and Events

When a major event is generated within apcupsd, control is passed to the script apccontrol normally found in /etc/apcupsd/apccontrol. The event name, and a number of other important parameters are passed to the script.

The major function of the apccontrol script is to performa a shutdown of the system (as well as the killpower operation). In addition, another major task for this script is to notify you by email when certain events such as powerfail occur.

Since apccontrol is a script, you can customize it to your own needs using any text editor. To do so, you must have a minimal knowledge of Unix shell programming. In addition, another feature is that you can write your own scripts that will be automatically called by apccontrol before any of its own code is executed. Details of the events and how to program them are contained in the Advanced topics section entitled Customizing Event Handling (see Customizing Event Handling).

7.3 hid-ups and USB Specific Information

The UPS has an internal set of timers and remaining capacity counters, which it uses to determine when to shutdown. These are in addition

to the apcupsd counters BATTERYLEVEL and MINUTES. As a consequence, apcupsd will shutdown on the first limit that triggers (either an apcupsd limit, or a UPS limit). The UPS internal counter equivalent to BATTERYLEVEL can be found in the hid-ups report as RemainingCapacityLimit, which is typically factory set to 10 percent. In addition, the Low Battery signal is normally given by the UPS when less than 2 minutes of run time remain.

7.4 apcupsd Network Monitoring (CGI) Programs

With this release, there are five CGI programs (multimon.cgi, multimoncss.cgi, upsstats.cgi, upsfstats.cgi, and upsimage.cgi). To have them properly installed, you must run the `./configure` command with `--enable-cgi` and you should specify an installation directory with `--with-cgi-bin=` or load them manually. To install the Cascading Style Sheet, which is used by multimoncss.cgi, you must use the `--with-css-dir=` option. The default directory for installation of the CGI programs is /etc/apcupsd, which is not really where you want them if you are going to use them. Normally, they should go in the cgi-bin of your Web server.

Once built and loaded, they will give you the status of your UPS or UPSes over the network.

Normally only `multimon.cgi` or multimoncss.cgiis directly invoked by the user. However, it is possible to directly invoke upsstats.cgi and upsfstats.cgi. upsimage.cgi should never be directly invoked as it is used by upsstats.cgi to produce the bar charts.

7.5 Setting up and Testing the CGI Programs

7.5.1 Network Information Server (NIS): Before using multimon and the other CGI programs, first ensure that apcupsd is configured to run the Network Information Server. This is done by setting **NETSERVER on** in /etc/apcupsd/apcupsd.conf. This switch is on by default. If you are unsure of its state, see the section at the end of this chapter concerning the Client test program.

Next you must edit the hosts file /etc/apcupsd/hosts.conf and at the end, add the name of the hosts you want to monitor and a label string for them. Kern Sibbald uses multimon.conf unmodified from what is on the source distribution. However, he has modified the hosts.conf file to contain the following three lines:

```
MONITOR matou "Server"
MONITOR polymatou "Backup server"
MONITOR deuter  "Disk server"
```

matou, polymatou, and deuter are the network names of the three machines currently running apcupsd. Please note that the network names may either be IP addresses or fully qualified domain names. The network name (or IP address) may optionally be followed by **:<port>**, where the port is the NIS port address you wish to use. This is useful if you are running multiple copies of apcupsd on the same system or if you are running in a mixed vendor environment where the NIS port assignments differ. An example could be the following:

```
MONITOR matou "Server"
MONITOR polymatou "Backup server"
MONITOR deuter  "Disk server"
MONITOR polymatou:7001 "APC USB UPS"
```

where the USB copy of apcupsd has been configured to use port 7001 (with `--with-nis-port=7001` on the `./configure` or by modifying apcupsd.conf). Note, the default NIS port is 3551 on most platforms.

To test multimon.cgi, you can execute it as non-root directly from the source cgi build directory. To do so, enter at a shell prompt:

```
./multimon.cgi
```

If everything is set up correctly, it will print a bunch of HTML with the values of the machines that you have put in the hosts.conf file. It should look something like the following (note, only a small portion of the output is reproduced here):

```
Content-type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
     "http://www.w3.org/TR/REC-html40/loose.dtd">
<HTML>
<HEAD><TITLE>Multimon: UPS Status Page</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<TABLE BGCOLOR="#50A0A0" ALIGN=CENTER>
```

```
<TR><TD>
<TABLE CELLPADDING=5>
<TR>
<TH COLSPAN=10 BGCOLOR="#60B0B0">
<FONT SIZE="+2">APCUPSD UPS Network Monitor</FONT>
<BR>Sun Jan 16 12:07:27 CET 2000</TH>
</TR>
<TR BGCOLOR="#60B0B0">
<TH COLSPAN=1>System</TH>
<TH COLSPAN=1>Model</TH>
<TH COLSPAN=1>Status</TH>
...
```

If you do not get similar output, check the permissions of the /etc/apcupsd directory and of those of /etc/apcupsd/hosts.conf to ensure that your web server can access it. At many sites such as mine, the Apache server is not running as root, so you must be careful to ensure that that /etc/apcupsd/hosts.conf and /etc/apcupsd/multimon.conf are world readable.

To invoke multimon in your Web browser, enter:

http://<your-site>/cgi-bin/multimon.cgi

You should get something similar to the screen shot shown below.

If you wish additional control over the colors, type faces, and sizes of the multimon output, you might wish to use multimoncss.cgi in place of multimon. In this case, you simply edit the multimon.css file to specify the styles you prefer. There are several sample Style Sheet files in the cgi subdirectory of the source tree.

To see a working example of the these programs, visit http://www.apcupsd.com/cgi-bin/multimon.cgi or http://www.apcupsd.com/cgi-bin/multimoncss.cgi

7.5.2 multimon.cgi: This program monitors multiple UPSes at the same time. A typical output of multimon.cgi as displayed in your Web browser might look like the following:

APCUPSD UPS Network Monitor
Thu Dec 02 17:08:27 CET 1999

System	Model	Status	Battery Chg	Utility	UPS Load	UPS Temp	Batt. Run Time	Data
Server	SMART-UPS 1000	ONLINE	100.0 %	228.8 VAC	6.2 %	35.1 °C	167.0 min.	All data
Backup server	SMART-UPS 1000	ONLINE	100.0 %	228.8 VAC	11.4 %	29.2 °C	112.0 min.	All data
Disk server	SMART-UPS 1000	ONLINE	100.0 %	230.1 VAC	6.2 %	35.1 °C	167.0 min.	All data

The machines monitored as well as the values and their column headings are all configurable (see /etc/apcupsd/hosts.conf and /etc/apcupsd/multimon.conf)

7.5.3 upsstats.cgi: By clicking on the **system** name in the **multimon.cgi** display, you will invoke upsstats.cgi for the specified system, which will produce a bar graph display of three of the monitored values. For example,

You can display different bar graphs by selecting different variables from the drop down menus at the top of each of the three bar graphs.

As with multimon, if you have your local host configured in the /etc/apcupsd/hosts.conf file, you can execute it from a Unix shell from the source cgi directory as follows:

7.5.4 ./upsstats.cgi: As with multimon, quite a few lines of html should then be displayed.

7.5.5 upsfstatus.cgi: If you would like to see all of the STATUS variables available over the network, click on the **Data** field of the desired system,

and your browser will display something like the following:

```
APC      : 001,048,1109
DATE     : Thu Dec 02 17:27:21 CET 1999
HOSTNAME : matou.sibbald.com
RELEASE  : 3.7.0-beta-1
CABLE    : Custom Cable Smart
MODEL    : SMART-UPS 1000
UPSMODE  : Stand Alone
UPSNAME  : UPS_IDEN
LINEV    : 223.6 Volts
MAXLINEV : 224.9 Volts
MINLINEV : 222.3 Volts
LINEFREQ : 50.0 Hz
OUTPUTV  : 223.6 Volts
LOADPCT  :   6.2 Percent Load Capacity
BATTV    : 27.9 Volts
BCHARGE  : 100.0 Percent
MBATTCHG : 5 Percent
TIMELEFT : 167.0 Minutes
MINTIMEL : 3 Minutes
SENSE    : High
DWAKE    : 060 Seconds
DSHUTD   : 020 Seconds
LOTRANS  : 196.0 Volts
HITRANS  : 253.0 Volts
RETPCT   : 050.0 Percent
STATFLAG : 0x08 Status Flag
STATUS   : ONLINE
ITEMP    : 35.1 C Internal
ALARMDEL : Low Battery
LASTXFER : U command or Self Test
SELFTEST : NO
STESTI   : 336
DLOWBATT : 02 Minutes
DIPSW    : 0x00 Dip Switch
REG1     : 0x00 Register 1
REG2     : 0x00 Register 2
REG3     : 0x00 Register 3
MANDATE  : 01/11/99
SERIALNO : GS9903001147
BATTDATE : 01/11/99
NOMOUTV  : 230.0
NOMBATTV :  24.0
HUMIDITY : N/A
AMBTEMP  : N/A
EXTBATTS : 0
BADBATTS : N/A
FIRMWARE : 60.11.I
APCMODEL : IWI
END APC  : Thu Dec 02 17:27:25 CET 1999
```

You should get pretty much the same output mixed in with html if you execute upsfstats.cgi directly from a Unix shell in the cgi subdirectory as explained above for upsstats.cgi and multimon.cgi.

7.5.6 Working Example: To see a working example of the above programs, visit http://www.apcupsd.com/cgi-bin/multimon.cgi.

7.5.7 Client Test Program: When your Network Information Server is up and running, you can test it using a simple program before attempting to access the server via your Web server. The test program is called client.c and can be found in the examples subdirectory of the source distribution. To build the program, when in the examples directory, use something like the following:

```
cc client.c ../lib/libapc.a -o client
```

Then execute it:

```
./client <host>[:<port>] [<command>]
```

Where **host** is the name of the host or the IP address of the host running the Network Information Server. The default is the local host. You may optionally specify a port address separated from the host name with a colon. You may also optionally specify a single command to be executed. If you specify a command, that command will be executed and the client program will exit. This is a very simple and useful way of pulling the **status** or **events** data into another program such as Perl.

If no error messages are printed, it has most likely established contact with your server. Anything that you type as standard input will be passed to the server, and anything the server sends back will be printed to standard output. There are currently two commands recognized by the server: **events** and **status**. Hence the following commands:

```
./client
status
events
xyz
^D
```

should produce the status listing (the same as produced by `apcaccess status`), followed by the list of the last 10 events (in response to the **events** command), and finally **Invalid command** in response to the **xyz** input, which is not a valid command. The control-D terminates the **client** program.

The purpose of this program is to show you how to write your own program that can determine the status of apcupsd and act any way you want (e.g. send you email messages on certain events like line voltage boost, ...).

7.5.8 A Tip from Carl Erhorn for Sun Systems: It is possible to run the CGI code to monitor your UPS using the answerbook HTTP server that runs on Solaris. As long as your server has the Answerbook2 web server installed and running, you can insert the cgi scripts into the cgi directory of the web server, and access the cgi using something like:

http://hostname:8888/cgi/multimon.cgi

7.5.9 Credits: Many thanks go to Russell Kroll <rkroll at exploits.org> who wrote the CGI programs to work with his UPS Monitoring system named Network UPS Tools (NUT). Thanks also to Jonathan Benson <jbenson at technologist.com> for initially adapting the upsstatus.cgi program to work with apcupsd.

We have enhanced the bar graph program and hope that our changes can be useful to the original author in his project.

7.5.10 Security Issues:

- **apcupsd** runs as root.
- If you have **NETSERVER ON** in your apcupsd.conf file (which is the deault), be aware that anyone on the network can read the status of your UPS. This may or may not pose a problem. If you don't consider this information privileged, as is the case for me, there is little risk. In addition, if you have a firewall between your servers and the Internet, crackers will not have access to your UPS information. Additionally, you can restrict who can access your apcupsd server by using inted to run the sservice and using access control lists with a TCP wrapper or by configuring TCP wrappers in apcupsd (see below for TCP Wrapper details).

- If you are running master/slave networking with a single UPS powering multiple machines, be aware that it is possible for someone to simulate the master and send a shutdown request to your slaves. The slaves do check that the network address of the machine claiming to be the master is that same as the address returned by DNS corresponding to the name of the master as specified in your configuration file.

7.5.10.1 TCP Wrappers

As of apcupsd version 3.8.2, TCP Wrappers are implemented if you turn them on when configuring (`./configure --with-libwrap`). With this code enabled, you may control who may access your apcupsd via TCP connections (the Network Information Server, and the Master/Slave code). This control is done by modifying the file: /etc/hosts.allow. This code is implemented but untested. If you use it, please send us some feedback.

7.6 Configuring Your EEPROM

If you have a SmartUPS, there are depending on the UPS at least 12 different values stored in the EEPROM that determine how the UPS reacts to various conditions such as high line voltage, low line voltage, power down grace periods, etc.

In general, for the moment, we do not recommend that you change your EEPROM values unless absolutely necessary. There have been several reported cases of problems setting the Low Transfer Voltage. Consequently, if at all possible, do not attempt to change this value.

If despite these warnings, you must change your EEPROM, we recommend connecting your UPS to a Windows or NT machine running PowerChute and making the changes.

7.6.1 apcupsd No Longer Configures EEPROM: Unlike version 3.8.6, apcupsd version 3.10.x no longer has code to program the EEPROM. Instead we have implemented interactive EEPROM modification code in the apctest program. EEPROM programming must be done with apcupsd stopped so that apctest can access the UPS. In addition, EEPROM programming is currently implemented only for UPSes using the Smart protocol running in serial mode. Perhaps at a later time when the appropriate kernel modifications are standard, we will extend EEPROM programming to USB models.

Before changing your EEPROM, you should make a printed copy of the

current state of your UPS before any EEPROM changes so that you can check the changes that you have made. Do so by printing a copy of the output from `apcaccess status` and also print a copy of the output from `apcaccess eprom`.

Once this is done, choose which values of the EEPROM you want to change. Typical output from apcaccess should look like the following:

```
apcaccess eeprom

Valid EPROM values for the SMART-UPS 1000

                       Config        Current  Permitted
Description              Directive     Value    Values
================================================================
Upper transfer voltage   HITRANSFER    253      253 264 271 280
Lower transfer voltage   LOTRANSFER    196      196 188 208 204
Return threshold         RETURNCHARGE  0        00 15 50 90
Output voltage on batts  OUTPUTVOLTS   230      230 240 220 225
Sensitivity              SENSITIVITY   H        H M L L
Low battery warning      LOWBATT       2        02 05 07 10
Shutdown grace delay     SLEEP         20       020 180 300 600
Alarm delay              BEEPSTATE     0        0 T L N
Wakeup delay             WAKEUP        0        000 060 180 300
Self test interval       SELFTEST      336      336 168 ON  OFF
```

where the Current Value will depend on how your UPS is configured, and the Permitted Values will depend on what UPS model you have.

7.6.2 Using apctest to Configure Your EEPROM: To make the EEPROM changes with apctest you must first stop the apcupsd daemon

apctest is not installed during the installation process, so to use it you will need to do the following after having built apcupsd:

```
cd <apcupsd-source>/src
su
<root-password>
./apctest
```

At that point, you should get output similar to the following:

```
2003-07-07 11:19:21 apctest 3.10.6 (07 July 2003) redhat
Checking configuration ...
Attached to driver: apcsmart
sharenet.type = DISABLE
cable.type = CUSTOM_SMART

You are using a SMART cable type, so I'm entering SMART test mode
mode.type = SMART
Setting up serial port ...
Creating serial port lock file ...
Hello, this is the apcupsd Cable Test program.
This part of apctest is for testing Smart UPSes.
Please select the function you want to perform.

1) Query the UPS for all known values
2) Perform a Battery Runtime Calibration
3) Abort Battery Calibration
4) Monitor Battery Calibration progress
5) Program EEPROM
6) Enter TTY mode communicating with UPS
7) Quit

Select function number:
```

You might want to run option 1) just to ensure that apctest is properly talking to your UPS. It will produce quite about 70 lines of output.

To program the EEPROM, select option 5), and you will get the EEPROM menu as follows:

```
This is the EEPROM programming section of apctest.
Please select the function you want to perform.

1) Print EEPROM values
2) Change Battery date
3) Change UPS name
4) Change sensitivity
5) Change alarm delay
6) Change low battery warning delay
7) Change wakeup delay
8) Change shutdown delay
9) Change low transfer voltage
10) Change high transfer voltage
11) Change battery return threshold percent
12) Change output voltage when on batteries
13) Change the self test interval
14) Set EEPROM with conf file values
15) Quit

Select function number:
```

If you wish to use the old pre-3.10.x method of EEPROM programming with values specified in the apcupsd.conf file, select option 14). However, we recommend that you start with item 1) to see what EEPROM values apctest finds. This command can take a few minutes to run, so be patient. The values printed should be the same as what you got using apcaccess, but in addition, the EEPROM battery date and UPS Name should be displayed. For example:

```
Select function number: 1

Doing prep_device() ...

Valid EEPROM values for the SMART-UPS 1000

                       Config        Current  Permitted
Description              Directive     Value    Values
=====================================================================
Upper transfer voltage   HITRANSFER    253      253 264 271 280
Lower transfer voltage   LOTRANSFER    196      196 188 208 204
Return threshold         RETURNCHARGE  0        00 15 50 90
Output voltage on batts  OUTPUTVOLTS   230      230 240 220 225
Sensitivity              SENSITIVITY   H        H M L L
Low battery warning      LOWBATT       2        02 05 07 10
Shutdown grace delay     SLEEP         20       020 180 300 600
Alarm delay              BEEPSTATE     0        0 T L N
Wakeup delay             WAKEUP        0        000 060 180 300
Self test interval       SELFTEST      336      336 168 ON  OFF
=====================================================================
Battery date: 07/31/99
UPS Name    : UPS_IDEN
```

At this point, you can select any item from 2) to 13) to modify the appropriate value. You will shown the existing value and prompted for the new values.

We recommend that you change the EEPROM as little as is absolutely necessary since it is a somewhat delicate process that has occasionally produced problems (i.e. improper EEPROM values are displayed after the update). Fortunately this seems to be quite rare and was much more likely to occur with the old "batch" like process especially if incorrect values were supplied.

8 Maintaining Your UPS

If you have your UPS long enough, you will probably have battery problems. Below, you will find some suggestions for replacing batteries. One

important note of caution: at least one user purchased one of the non-APC batteries noted below and found out that they would not fit into his unit. This required cutting and soldering and other very undesirable things, so be extremely careful in measuring the batteries including every millimeter of the terminal connections which can cause problems.

Although you can do a hot swap of your batteries while the computer is running, it may not be very satisfactory because the unit will not know that the batteries have been swapped and apcupsd will continue to show Low Battery. To correct this situation, you must do a discharge and recharge of the battery followed by a battery recalibration using apctest. At that point the battery should be calibrated better. As noted below, Carl has found that it takes several discharge/charges before the runtime calibration is accurate. Take care not to discharge your battery too much as it tends to shorten the battery life.

8.1 What Various People Have to Say about Batteries

Here is what John Walker has to say about APC UPS batteries:

> I thought I'd pass on some information I've obtained which you'll probably eventually need. Besides, by writing it down I'll be able to find it the next time. I started installing mine in 1995-1996. Lead-acid batteries have a finite life even if not subjected to deep discharge cycles. For the batteries used by APC, this is typically four to six years. As part of the self-test cycle, the UPS measures the voltage of the battery at full charge (which falls as the battery ages), and if it's below about 90% of the value for a new battery, it sets off the "Replace battery" alarm, which it repeats every day. [on apcupsd versions prior to 3.8.0, this message is sent once, on version 3.8.0, it is sent every 9 hours - KES]. You will occasionally get a false alarm. It's a good idea if you get an alarm to repeat the self-test the next day and see if the alarm goes away. If the alarm is persistent, you need to replace the batteries, which can be done without powering down the UPS or load-you just open up the battery door, take out the old batteries, and hook up the new ones.
>
> APC makes "Replacement Battery Units" for each of the SmartUPS models, but they sell them directly only in the U.S.
>
> It's best to wait until the low battery alarm before ordering a replacement-keeping batteries on the shelf reduces their life unless you keep them fully charged.

And Andre Hendrick says:

> [For replacement batteries] You need to goto you your local Yamaha SeaDoo shop. There are 35 AMP Hour deep cycle marine batteries that are direct replacements. These are gel-cel and will double the runtime and/or cut your recharge time in half.

```
Jet Works
1587 Monrovia Ave.
Newport Beach CA 9266?
Tel: +1 714 548-5259

J-W Batteries, Inc.
Tel: +1 714 548-4017

WPS 49-1200
GEL-CELL KB-35 BATTERY
```

> For those that do not know what this means..... I found the best battery for APCC UPS products that use In the two systems below:

```
SMART-UPS 3000 10.9% is running at 327W runs for 47.0 min.
Smart-UPS 1250 22.3% is running at 279W runs for 54.0 min.

APCUPSD UPS Network Monitor
Thu Jan 18 21:55:36 PST 2001
System Model Status Battery Chg Utility UPS Load UPS Temp Batt. Run Time Data
Linux ATA Development SMART-UPS 3000 ONLINE
                    100.0 % 120.2 VAC 10.9 % 36.9 C 47.0 min. All data
Linux ATA Development II APC Smart-UPS 1250 ONLINE
                    100.0 % 119.6 VAC 22.3 % 45.9 C 54.0 min. All data
```

> Look at the numbers and see that these batteries are better and have more total running energy than standard ones.

```
SMART-UPS 3000 10.9% is running at 327W runs for 47.0 min.
Smart-UPS 1250 22.3% is running at 279W runs for 54.0 min.

APCUPSD UPS Network Monitor
Thu Jan 18 22:00:45 PST 2001
System Model Status Battery Chg Utility UPS Load UPS Temp Batt. Run Time Data
Linux ATA Development SMART-UPS 3000 ONLINE
                    100.0 % 120.2 VAC 19.2 % 36.9 C 27.0 min. All data
```

```
Linux ATA Development II APC Smart-UPS 1250 ONLINE
                    100.0 % 119.6 VAC 21.8 % 45.9 C 55.0 min. All data

SMART-UPS 3000 19.2% is running at 576W runs for 27.0 min.
Smart-UPS 1250 21.8% is running at 273W runs for 55.0 min.
Smart-UPS 1250 46.1% is running at 576W runs for 26.0 min.
```

> Kind of cool.
>
> The 1250 can outrun the 3000 by a factor of two under identical percentages, or run head to head for the same time.
>
> SMART-UPS 3000 is a 48V based or 4 batteries. Smart-UPS 1250 is a 24V based or 2 batteries.
>
> Cheers,
>
> ```
> Andre Hedrick
> Linux ATA Development
> ```

Finally, here is what Carl Erhorn has to say about batteries:

> Hi, Folks.
>
> Well, Kern was absolutely right. The problem with my UPS was batteries. It was unexpected though, because there was no indication of a bad battery right up until the UPS failed entirely.
>
> For those who might encounter the same thing, and don't know what's happening (I didn't either), here's what happened.
>
> A week or so ago, I turned on one of my SmartUPS 700-NET models. The load is a small dual P-III unix server (Solaris 8, X86) and a 4MM tape drive. During the normal selftest that runs when you first turn on any APC UPS, the UPS 'freaked out'. The alarm stuttered at about 4 or 5 beeps per second, and all the panel lights flashed spasmodically, as if something was loose inside the UPS.
>
> I turned off the UPS and it's load, then turned the UPS on again. This time, everything seemed fine. I booted the system that was attached, and there were no problems. The status monitor showed 9 minutes runtime (which indicates fairly low capacity), but the batteries showed fully charged. I began to suspect a bad inverter in the UPS.

However, Kern told me that he suspected the batteries. So I took the UPS offline, put an old SU-600 in it's place (just barely big enough to handle the startup peaks - I get an 'overload' lamp lit for about 2 seconds during boot), and checked out the batteries. They did indicate that they were near the end of life, so I ordered a replacement set. Those came in on Friday, and after the initial charge, a complete charge/discharge cycle to recalibrate the UPS, and some testing, I put it back in service.

Surprise! (Or maybe not?) Kern was right - there is nothing wrong with the inverter or the charging circuit, and the new cells fixed everything.

What confused me is that there was no 'replace battery' indication from the UPS, even when it failed, plus a fair amount of runtime indicated with a full charge. So if you see such behavior on one of your UPS models, it makes sense to replace the batteries, even if there is no indication that the batteries have failed yet.

One of the things I learned during this process is that the UPS internal calibration will lose accuracy over the life of the battery. I always do a recalibrate when I install new cells, but rarely do it after that, as it's time-consuming, and you really can't use the system attached to the UPS while doing it. Since my systems are almost constantly in use, it's a pain to schedule a recal, and I tend to put it off. This time it bit me. I'd suggest that folks do a recal at least once every six months. It will make your runtime estimates much more accurate, and also allows you to keep track of the state of your batteries.

For those who don't know how to do this, here's what you do. This proceedure should not be confused with the 'Recalibrate' feature in the APC PowerchutePlus software. They do not do the same thing.

>From APC's web site:

Perform a Runtime Calibration. This is a manual procedure and should not be confused with the runtime calibration performed through PowerChute plus. The batteries inside of the Smart-UPS are controlled by a microprocessor within the UPS. Sometimes it is necessary to reset this microprocessor, especially after the installation of new batteries. Stop the PowerChute plus software from running and disconnect the serial cable. There must be at least a 30% load attached to the UPS during this procedure, but the process will cause the UPS to shut off and cut power to its outlets. Therefore, attach a non-critical load to the

UPS and then force the UPS on battery by disconnecting it from utility power. Allow the unit to run on battery until it turns off completely. Make sure a 30% load is present! Plug the UPS back into the wall outlet and allow it to recharge (it will recharge more quickly turned off and with no load present). Once the unit has recharged, the "runtime remaining" calculation should be more accurate. Remember that if the unit is an older model, then the runtime will not improve significantly.

Background:

An APC Smart-UPS has a microprocessor which calculates runtime primarily based on the load attached to the UPS and on its battery capacity. On the right side of the front display panel there is a vertical graph of five LEDs. Each LED is an indication of battery charge in increments of twenty percent: 20, 40, 60, 80, 100% (bottom to top). For example, if the battery charge is 99%, then only four of the five LEDs are illuminated.

To ensure that an operating system receives a graceful shutdown when using PowerChute plus or a SmartSlot accessory, an alert is generated by the Smart-UPS indicating that the UPS has reached a low battery condition. The alert is audible (rapid beeping), visual (flashing battery LED or LEDs), and readable through the graphical interface of PowerChute plus software (or a native UPS shutdown program within a particular operating system.) In order to calculate this "low battery condition," all Smart-UPS products have a preconfigured low battery signal warning time of two minutes (this is the factory default setting). There are a total of four user-changeable settings: 2, 5, 7, or 10 minutes. If the low battery signal warning time is set for 2 minutes, then the alerts will activate simultaneously two minutes prior to shutdown. Similarly, if the total runtime for a particular UPS is 30 minutes with a low battery signal warning time set at 10 minutes, then the UPS will run on battery for 20 minutes before the low battery alert begins.

Total runtime is primarily based on two factors, battery capacity and UPS load. UPS load and runtime on battery are inversely proportional: as load increases, battery runtime decreases and vice versa. When utility power is lost, the UPS begins discharging the battery in order to support the attached load. Once power returns, the Smart-UPS will automatically begin to recharge its battery.

My comments on this proceedure:

I believe this proceedure works for all APC models that calulate

runtime, not just the SmartUPS. It's important that you load the UPS to 30% of the UPS capacity, as reported by apcupsd or another UPS monitor program. I've found that normal house lamps of different wattages allow me to adjust the load to almost exactly what I want, which is between 30% and 35% of the UPS capacity. This is critical te getting an accurate reading (according to the APC web documents). Always bring the UPS to 100% charge first, as indicated by the front panel lamps, or your UPS monitoring software.

Set the UPS shutdown time to 2 minutes, all other settings to nominal, and disconnect the serial port cable from the UPS before running the recalibration. If you leave a monitoring program running through the serial port, it will turn the UPS off early, and you don't want to do that during a recalibration run. When the run is complete, and the UPS turns off, you can reattach the serial cable, and the normal loads, and recharge the batteries normally. If you think you might have a power outage during the recharge time, allow the UPS to recharge to 20% or so (indicated by the panel lamps) before trying to use the computer system. This will allow the UPS to handle short dropouts while it recharges. Of course, if you can leave the computer off during the recharge time, the UPS will recharge much faster.

As an aside, when the batteries failed, my total runtime at 100% charge and an idle state was 9 minutes, which is pretty bad. I replaced the batteries with extended capacity cells, which add about 15% to the stock capacity. Now, after two complete charge/ discharge cycles, 100% charge shows the available runtime to be 42 minutes on the system when it's idle, and 33 minutes when the system is very busy. The differences are due to the load of the computer, when the disks are busy, and the cpus are not in a halted state (my system halts the cpus when they are idle, to save power and lower heat, as do other OS like Linux), when compared to an idle state. Apcupsd indicates the load is about 27% when idle, and as much as 37% when heavily loaded.

I've found that two charge/discharge cycles result in a more accurate recalibration when installing new cells. It appears that some batteries need to be put through a couple of complete cycles before they reach their full capacity. I've also noticed that the full-charge voltage is different for each battery until they have been through two cycles. On the initial charge of my new batteries, the 100% charge voltage on the two cells was almost .5 VDC apart. After two complete cycles, the batteries measure within .01 VDC of each other!

I hope this information helps anyone who might encounter the problem I saw, and also shows folks how to recal their batteries. If you haven't done a complete recalibration in a year or two, I'd recommend it, so that you have warning of a low battery instead of what happened to me.

Regards,

—Carl

8.2 Where Carl Suggests You Get Batteries

Hi, Folks.

I'm just replacing the batteries in one of my SmartUPS models, and it occurs to me that some of you may not know about the place I get them from. I have no relationship with this company, other than as a customer, but I feel they know what they are doing, their prices are fair, and they have some interesting batteries available that you can't obtain from APC.

These are the reasons I use them, and I thought this information might be useful to the US list members. They will ship outside of the US. If you have questions, you can contact them through the email address listed on their web pages. They have always responded pretty quickly to my questions.

The company is called Battery Wholesale Distributors, and they are located in Georgetown, Texas. If you have questions, you can reach them by phone at (800) 365-8444, 9:00AM to 5:00PM (their local time), Monday through Friday. I've gotten email from them on the weekends, although the office is not open then.

I won't post prices, as you can get current pricing from their web site. They have an entire section dedicated to APC replacement batteries, and it's easy to find what you need. You can order over the web, or by phone. They accept all the usual credit cards.

The web site (as you might guess) is: www.batterywholesale.com

The thing I really like is that they have found manufacturers who make batteries in the standard case sizes, but have additional capacity over the original batteries shipped with the APC UPS models. Often, the difference is as much as 15% or so, and this can result in additional runtime. It's a nice upgrade for a minor increase in price.

They are also 'green-aware', in that they encourage you to recycle your old batteries, and will accept the old batteries back from you if you cannot find a local place that recycles them. You pay the shipping, but I think other than that, there is no charge. I've never done this, as I have a battery retailer just down the street who will accept my old batteries.

Anyway, if you didn't know about these folks, put the info aside where you can find it when you need replacement batteries. I won't make any guarantees, but I've been very pleased with their products, service, and pricing. I hope you find them as helpful to you as I do. I've been dealing with them since about 1994, and have never been disappointed. The owner of the place also is very good on technical issues, so if you have questions on their products, he can get as technical as you need to go.

```
Regards,
--Carl
```

Here is a link to the APC Battery Store.

9 Frequently-Asked Questions

See the bugs section of this document for a list of known bugs and solutions.

Q: Why all the craziness with custom serial cables?

A: It was nothing more nor less than a form of customer control. For a long time APC wanted to keep other people from talking to its UPSes so it could lock out potential competition for its PowerChute software. Scrambling the leads on its serial cables was a cheap way to accomplish this – in fact, they tended to be wired so that if you tried a straight-through cable, opening a serial link to the UPS would be interpreted as a shutdown command!

(Hardware companies often think like this – they lock up interfaces by instinct, cornering a small market rather than growing a bigger one. It's fundamentally stupid and self-defeating, but it's the kind of stupid that tends to sound good at an executive meeting.)

Fortunately, APC has lost a lot of this attitude since about 2000; nowadays they even release technical information to the apcupsd maintainers.

Q: What UPS brands does apcupsd support?

A: Currently apcupsd supports only APC UPSes. However, some companies such as Hewlett Packard put their own brand name on APC manufactured UPSes. Thus even if you do not have an APC branded UPS, it may work with apcupsd. You will need to know the corresponding APC model number. apcupsd supports all the popular APC models. See the installation and configurations sections of this document for more details.

Q: Does apcupsd support Windows?

A: With release 3.8.0, apcupsd now runs on Win95/98, WinMe, WinNT, and Win2000 machines. All features of the Unix versions of apcupsd are implemented. The UPS EEPROM programming features of apcupsd have not been tested under Windows. Version 3.8.0 does not support simple signaling UPSes (BackUPS, etc). Version 3.8.1 does support most simple signaling UPSes, but not all cables (due to deficiencies in the Windows serial port API). Please note that we have had reports that apcupsd does not work properly on the WinXP system. If you have any information on this, please email us.

Q: I don't have a cable, which one should I build?

A: First you must know if you have an apcsmart UPS or a voltage-signalling UPS. See the table of supported UPSes (see type_table). If you have a apcsmart UPS, we recommend building a Custom Smart (see Smart-Custom Cable for SmartUPSes) cable. If you have a voltage-signaling UPS, we recommend that you build a Custom Simple (see

Voltage-Signalling Cable for "dumb" UPSes) cable.

Q: How much CPU resources does apcupsd use?

A: Depending on your CPU speed, you may see more or less of the CPU consumed by apcupsd. On a 400MHz Unix system, the CPU usage should fall well below 0.1%. On slower systems, the percentage will increase proportionally to the decrease in the CPU speed. On a 400Mhz Win98 machine, the CPU usage will be on the order of 0.5-1.0%. This is higher than for Unix systems. However, compared to the 30% CPU usage by APC's PowerChute (the version on the CDROM shipped with my UPS), apcupsd's 0.5-1.0% is very modest.

If you configure apcupsd to run with pthreads (`--with-pthreads` on the `./configure` line), apcupsd will run considerably faster, otherwise said, it will consume less of your CPU, and it will use approximately one third of the memory. For example, Carl Erhorn reports that on his Solaris system, "With the old 3-process version, we averaged about

4.8MB of total memory used. With the new single process, we use only about 1.7MB! That's also a very good improvement."

Q: What language is apcupsd written in?

A: It is written entirely in C.

Q: We are using apcupsd-3.8.1-1 in RedHat 6.2. The slave, when shutting down, is reporting an error at line 436 of apcupsd.c. The error is initiated by `apcupsd --killpower`! What can we do to fix this, and is it critical?

A: No, the error is not serious. Unfortunately, the documentation in the area of master/slaves is not very detailed, and for that reason, your slave setup is not totally correct as explained below.

On master machines, we modify /etc/rc.d/init.d/halt to re-invoke apcupsd with the `--killpower` option (actually the script apccontrol is called). This causes the UPS to send the codes to the UPS to make it power off.

On slave machines, these modifications should not be made to the /etc/rc.d/init.d/halt script since the slave has no connection to the UPS.

To eliminate the problem, on all your slave machines, either restore the original halt file, or simply delete all the lines containing ***apcupsd***, which were inserted by the apcupsd installation process.

Q: To test apcupsd, I unplugged the UPS to simulate a power outage. After the machine went into the shutdown process I plugged the UPS back into the commercial power source. This caused the shutdown process to hang after the daemon tried to shut-off the ups. Have you run into this problem, and if so do you have a remedy?

A: Normally, once the shutdown process has begun, we cannot stop it, though there *is* some code that tries to do so, we don't consider it a very good idea – how do you stop a shutdown that has killed off half of the daemons running on your system? Most likely you will be left with an unusable system. In addition, when apcupsd is re-executed in the halt script after the disks are synced, it tries to shut off the UPS power, but the UPS will generally refuse to do so if the AC power is on. Since we cannot be 100% sure whether or not the UPS will shut off the power, we don't attempt to reboot the system if we detect that the power is back as it might then get caught by a delayed power off (at least for Smart UPSes).

Q: After running apcupsd for a while, I get the following error: "Serial communications with UPS lost." What is the problem?

A: We use standard Unix serial port read() and write() calls so once a connection is made, we generally have few problems. However, there have been reports that APC's SNMP Management Card can cause serial port problems. If you have such a card, we suggest that you remove it and see if the problem goes away. It is also possible that some other process such as a getty is reading the serial port.

Q: When apcupsd starts, I get the following error: "attach_shmarea: cannot get shm area: Identifier removed." What is the problem?

A: This problem and the problem of **cannot create shm area** are due to the fact that the shared memory key that apcupsd wants to use is already in use. This happens most frequently when there is an old zombie apcupsd process still in the system. The solution is to remove the old process. You can often see what is going on by doing a: `ipcs` command as root when apcupsd is not running. If you see a segment with the key 0x10feed01, you can be sure there is some old apcupsd process still using it. If you cannot kill the old process, you can try using `ipcrm` (see the man pages). Recent versions of apcupsd starting with apcupsd-3.8.2Beta6 should no longer have this problem as they will automatically try using a different key.

Q: I get the following error: "Starting apcupsd power management. Mar 20 21:19:40 box apcupsd[297]: apcupsd FATAL ERROR in apcserial.c at line 83. Cannot open UPS tty /dev/cua01: No such file or directory." What is the problem?

A: The two most likely causes of your problem are: 1. You have the wrong serial port device name in the apcupsd.conf file. 2. The device name is not defined on your system. Suggestions for proceeding:For the first item, check what your serial port device should be named. You might be able to find the name with an:

```
ls /dev
```

Normally there will be hundreds or even thousands of names that print. If that doesn't produce anything useful, you can try step 2. Perhaps your device is not defined. To get more information on your devices try man `MAKEDEV` or `find / -name MAKEDEV`. It is often located in `/dev/MAKEDEV`. Looking at the documentation may tell you what the correct name is, or at least allow you to create the device.

Q: How do I ensure that the slaves shutdown before the master?

A: There are several strategies for getting the slaves properly shutdown before shutting down the master. The first is to make the master wait a period of time for the slaves to shutdown before doing its own shutdown. Currently, the master always waits 30 seconds before starting its own shutdown. If this is insufficient, you can add additional time by putting an appropriate **sleep** shell command in the /etc/apcupsd/apccontrol file just before the actual system shutdown command is executed (there are something like 3 places). The second strategy is to put a TIMEOUT value in the apcupsd.conf file on the slave that is sufficiently short that you are sure that the slave will shutdown before the master. If the shutdown is done with a poweroff, this will also save power so that the master can stay up longer.

Q: How do I ensure that my database server is correctly shutdown?

A: You simply add whatever commands are necessary in the appropriate case statements in /etc/apcupsd/apccontrol, which is a standard script file that is called to actually do the shutdown. Alternatively, you can add your own script file that will be called before doing the commands in apccontrol. Your script file must have the same name as the appropriate case statement in apccontrol; it must be executable; and it must be in the same directory as apccontrol.

Q: I have Win2k Advanced server, and when starting the service, get: Could not start the Apcupsd Server service on Local Computer. Error 1067: The process terminated unexpectedly

A: The most common error causing your problem is an incorrect serial port specification on your DEVICE directive. It should be:

```
DEVICE /dev/com2
```

On WinNT machines, and probably Win2000 machines you MUST use /dev/com2 unless you modify the behavior of the boot process to prevent Windows from probing the port. This is documented in our manual for WinNT. Although I imagine it is the same for Win2000, I am not sure.

The second most common problem is bad placement of the files i.e. you did not install them in c:\apcupsd Unfortunately for the current release, this path is "hard coded" into the binaries.

The third most common problem is that you did not run the setup.bat script after loading the files. This is necessary to install apcupsd as a service.

If all the above fails, try starting apcupsd by hand inside a CYGWIN rxvt window if you use an rxvt window rather than a DOS window, you will see many more of the error messages.

In addition, most of the apcupsd startup errors are reported in: c:\apcupsd\etc\apcupsd\apcupsd.events

Many error messages associated with Windows services will be reported in the Windows System Log.

Q: When using USB, I get the following log messages: usb-uhci.c: interrupt, status 3, frame# 826. What does it mean?

A: It means one transfer worked (bit 0 in status) and another one (after that) failed (bit 1) at time frame 826. This kind of soft error is common on USB and if everything seems to be working, you can ignore it.

Q: apcnisd doesn't work. It always gives: FATAL ERROR in apcipc.c at line 497. attach_shmarea: shared memory version mismatch (or UPS not yet ready to report)

A: Unfortunately apcnisd does not work with pthreads enabled. You have the following options:

1. If you build with pthreads enabled, apcnisd will not work no matter what you do.
2. If you build with pthreads enabled, and you want to have network information from apcupsd, you must set NETSERVER ON. This is the configuration we recommend (i.e. using pthreads and NETSERVER ON).
3. If you build with pthreads disabled, you have the choice of using apcnisd or the NETSERVER code. If you wish to use apcnisd, you must set NETSERVER OFF
4. If you build with pthreads disabled, and you do not use apcnisd, you must set NETSERVER ON if you wish to have network information from apcupsd.

Concerning the names one sees with "ps".

1. With pthreads enabled, on Linux machines, you will see multiple copies of apcupsd running, but they will all be called apcupsd rather than apcmain, apcser, ... They will still run as LWP, but we are unable to set the names on threads (LWP). Note, though

ps shows "multiple copies" of apcupsd running, it is really one memory image but with multiple threads.

2. With pthreads disabled, we are able to set the child process names (at least on Linux) so you will see apcmain, apcser, apcnis, ... in the ps output. In this case, they are really different processes each with its own memory image (the code image is most likely shared).

10 Apcupsd Bugs

Unfortunately, it seems that every program has some bugs. We do our best to keep the bugs to a minimum by extensive testing. However, because of our inherent nature to occasionally overlook things and the fact that we don't have all the UPS models nor the APC documentation on those models, apcupsd will have some bugs.

As the bugs become known to us, we will post them on the bug tracking system at SourceForge.

Advanced topics

11 Customizing Event Handling

When apcupsd detects anomalies from your UPS device, it will make some decisions that usually result in one or more calls to the script located in /etc/apcupsd/apccontrol. The **apccontrol** file is a shell script that acts on the first argument that apcupsd passes to it. These actions are set up by default to sane behavior for all psituations apcupsd is likely to detect from the UPS. However, you can change the apccontrol behavior for every single action.

To customize, so create a file with the same name as the action, which is passed as a command line argument. Put your script in the /etc/apcupsd directory.

These events are sent to the system log, optionally sent to the temporary events file (/etc/apcupsd/apcupsd.events), and they also generate a call to /etc/apcupsd/apccontrol which in turn will call any scripts you have placed in the /etc/apcupsd directory.

Normally, /etc/apcupsd/acpcontrol is called only by apcupsd. Consequently, you should not invoke it directly. However, it is important to

understand how it functions, and in some cases, you may want to change the messages that it prints using `wall`. We recommend that you do so by writing your own script to be invoked by `apccontrol` rather than by modifying apccontrol directly. This makes it easier for you to upgrade to the next version of apcupsd

In other case, you may want to write your own shell scripts that will be invoked by apccontrol. For example, when a power fail occurs, you may want to send an email message to root. At present the arguments that apccontrol recognizes are:

When apcupsd detects an event, it calls the apccontrol script with four arguments as:

apccontrol <event> <ups-name> <connected> <powered>

where:

event is the event that occurred and it may be any one of the values described in the next section.

ups-name is the name of the UPS as specified in the configuration file (not the name in the EEPROM). For version 3.8.2, this is always set to **Default**

connected is 1 if apcupsd is connected to the UPS via a serial port (or a USB port). In most configurations, this will be the case. In the case of a Slave machine where apcupsd is not directly connected to the UPS, this value will be 0.

powered is 1 if the computer on which apcupsd is running is powered by the UPS and 0 if not. At the moment, this value is unimplemented and always 0.

11.1 apccontrol Command Line Options

apccontrol accepts the following command line options:

annoyme When a shutdown is scheduled, and the time specified on the ANNOYME directive in the apcupsd.conf file expires, this event is generated.

Default — does a `printf ‘‘Power problems please logoff.’’ | wall` then exits.

changeme When apcupsd detects that the mains are on, but the battery is not functioning correctly, this event is generated. It is repeated every x hours.

Default — does a `printf ‘‘Emergency! UPS batteries have failed\nChange them NOW’’ | wall` then exits.

commfailure This event is generated each time the communications line with the computer is severed. This event is not detected on dumb signaling UPSes.

Default -does a `printf ‘‘Warning serial port communications with UPS lost.’’ | wall` then exits.

commok After a commfailure event is issued, when the communications to the computer is re-established, this event will be generated.

Default — does a `printf ‘‘Serial communications with UPS restored.’’ | wall` then exits.

doreboot This event is depreciated and should not be used.

Default - does a reboot of the system by calling `shutdown -h now`

doshutdown When the UPS is running on batteries and one of the limits expires (time, run, load), this event is generated to cause the machine to shutdown.

Default does a shutdown of the system by calling `shutdown -h now`

emergency Does an emergency shutdown of the system by calling `shutdown -h now`

failing This event is generated when the UPS is running on batteries and the battery power is exhausted. The event following this one will be a shutdown.

Default — does a `printf ‘‘UPS battery power exhausted. Doing shutdown.\n’’ | wall` then exits.

loadlimit This event is generated when the battery charge is below the low limit specified in the apcupsd.conf file.

Default — does a `printf ‘‘UPS battery discharge limit reached. Doing shutdown.\n’’ | wall` then exits. After completing this event, apcupsd will immediately initiate a `doshutdown` event.

mainsback This event is generated when the mains power returns after a powerout condition. The shutdown event may or may not have been generated depending on the paramaters you have defined and

the length of the power outage. A cancel of a shutdown should never be attempted as it is very unlikely to succeed and will almost surely leave your machine in a indeterminate state.

Default — attempts to cancel the shutdown with a `shutdown -c` (not sure about that!!!!)

onbattery This event is generated 5 or 6 seconds after an initial powerfailure is detected. It means that apcupsd definitely considers the UPS to be on batteries. The onset of this event can be delayed by the ONBATTERYDELAY apcupsd.conf configuration directive.

Default — does a `printf ‘‘Power failure. Running on UPS batteries.’’ | wall` then exits.

offbattery This event is generated when the mains return only if the onbattery event has been generated.

Default — does nothing.

powerout This event is generated immediately when apcupsd detects that the UPS has switched to batteries. It may be due to a short powerfailure, an automatic selftest of the UPS, or a longer powerfailure. In many cases, you may want to inhibit the normal message sent/emailed by this event to avoid being annoyed by short power failures.

Default — does a `printf ‘‘Warning power loss detected.’’ | wall` then exits.

remotedown This event is generated on a slave machine when it detects either that the master has shutdown, or that a onbattery situation exists and the communications line has been severed. Despite the name, you should never reboot the machine — instead always shut it down.

Does a `shutdown -h now`

restartme This event is depreciated and should not be used.

Terminates the currently running apcupsd and then restarts it.

runlimit This event is generated when the MINUTES value defined in the apcupsd.conf file expires while in a power fail condition. The MINUTES is the remaining runtime as internally calculated by the UPS and monitored by apcuspd.

Does a `printf ‘‘UPS battery runtime percent reached. Doing shutdown.\n’’ | wall` then exits. After completing this event, apcupsd will immediately initiate a `doshutdown` event.

timeout This event is generated when the TIMOUT value defined in the apcupsd.conf file expires while in a power fail condition. It indicates that the total time in a power failure has been exeeded and the machine should be shutdown. Normally, with smart UPSes, this value is not used, but rather one relies on the remaining runtime (MINUTES) or the battery level (BATTERYLEVEL) values specified in the conf file.

Does a `printf ''UPS battery runtime limit exceeded. Doing shutdown.\n'' | wall` then exits. After completing this event, apcupsd will immediately initiate a `doshutdown` event.

startselftest This event is generated when apcupsd detects a self test by the UPS. Normally due to the 6 second onbattery delay default time, self test events are not detected.

This is called when apcupsd detects that the UPS is doing a self test. No action is taken.

endselftest This event is generated when the end of a self test is detected.

This is called when apcupsd determines that a self test has been completed. No action is taken.

mastertimeout This event is generated when a slave detects that a master has not contacted it in a reasonable time, or when a slave polls a master and gets no response in 30 seconds. This event applies only to the old master/slave networking code and not to the NIS server/slave mode.

No action is taken.

masterconnect This event is generated when the slave and the master reconnect. This event applies only to the old master/slave networking code and not to the NIS server/slave mode.

No action is taken.

To write your own routine for the **powerout** action, you create shell script named **powerout** and put it in the lib directory (normally /etc/apcupsd). When the **powerout** action is invoked by apcupsd, apccontrol will first give control to your script. If you want apccontrol to continue with the default action, simply exit your script with an exit status of zero. If you do not want apccontrol to continue with the default action, your script should exit with the special exit code of **99**. However, in this case, please be aware that you must ensure proper shutdown of your machine if necessary.

Some sample scripts (onbattery and mainsback) that email power failure messages can be found in the examples directory of the source code.

12 Master/Slave Configurations

If you have two or more computers that are powered by the same UPS and they are connected by a network, you can configure apcupsd so that the computer that controls the UPS (connected by the serial port or USB port), which is called the master, can provide information to other machines powered by the UPS, called slaves. When the master detects a power failure, it will notify all the slaves (maximum of twenty). If the master detects that the battery is low, it will also notify the slave so that the slave may perform a shutdown.

In addition, in cases where you wish to keep the master up longer than the slave, you can configure the slave to shut down in a predetermined time after the UPS goes on batteries.

If a picture is worth a thousand words for you, please see Configuration types..

12.1 Configuration Directives

If you are setting up a master/slave configuration, you will be required to make some modifications to the apcupsd.conf files after the build is done.

The minimum set of configuration directive changes needed to create a proper master and slave configuration files is described in the Configuration Examples section of this manual.

The details of these directives are explained in the Configuration Directives for Sharing a UPS section of the Configuration chapter of this document.

In addition, sample master and slave configuration files can be found in the <src>/examples directory (apcupsd.master.conf and apcupsd.slave.conf).

12.2 Master/Slave Problems

12.2.1 Master/Slave Shutdown: For additional details of shutting down a master/slave configuration, please see the Master/Slave Shutdown section of the Shutdown chapter (see Shutdown Sequence <1>) of the Technical Reference.

12.2.2 Server/Slave Networking using NIS and the NET Driver: It is also possible to implement a network of NIS server/slave apcupsds using the new 3.10.x code and the **net** driver. This mode of NIS server/slave networking is considerably different from the old method described at the beginning of this chapter. In the old code, there is a lot of configuration on both the master and slave side, and the master polls or sends info to the slave. Using the **net** driver is much simpler. However, you should carefully check that the NIS slave does a proper shutdown. In the master/slave code, the master ensures the best it can that the slave is shutdown or notified before it shuts down itself. On the other hand, using the net driver, the NIS server knows nothing about the NIS slaves that may be listening and thus takes no special precautions to ensure that the NIS slaves receive the shutdown signal. Since the NIS slave reads the master's data once per second there should be no shutdown problems, and our experience confirms this. This question can only be answered by carefully testing the shutdown.

In this NIS server/slave mode, the NIS server is a standard stand alone configuration except that it must have **NETSERVER on** in the configuration file and have an **NISPORT nnn** defined. Thus any apcupsd running in this mode then becomes the NIS server.

The NIS slave then uses the net driver to connect to the server's NIS output. In this mode, the NIS slave decides how often to poll the server for the NIS information. The NIS slave's conf file has **UPSTYPE net**, which will invoke the "network" driver. By setting this machine's **DEVICE** to be **server-ip:server-NIS-port** it will automatically connect to the NIS server and use the server's signals to shutdown the computer. In the example net slave configuration file below, the slave uses the NIS information provided by the computer **tibs** on port **3551**.

```
## apcupsd.conf v1.1 ##
UPSCABLE ether
UPSTYPE net
# Specify the server name:port where NIS is running
DEVICE tibs:3551
LOCKFILE /var/lock
BATTERYLEVEL 5
MINUTES 3
TIMEOUT 0
ANNOY 300
ANNOYDELAY 60
NOLOGON disable
EVENTSFILE /etc/apcupsd/apcupsd.events
UPSCLASS standalone
UPSMODE disable
#
```

```
# Use this to control the poll time.
#  the default is 60 or 1 minute
#
NETTIME 30
```

12.3 Network Problems with Master/Slave or Server/Slave Configurations

When working with a master/slave or server/slave configurations (one UPS powering more than one computer), the master/server and slave communicate via the network. In many configurations, apcupsd is started before the network is initialized. In this case, it is possible that the master will be unable to contact the slave. On apcupsd versions prior to 3.8.0, this could cause apcupsd to error off. The solution to this problem is to either force apcupsd to be started after the network and the DNS (fiddle the symbolic links in /etc/rc.d), or put the names of the slave machines in your /etc/hosts file, or even more preferable, use IP addresses rather than machine names. On some configurations, you may need to use fully qualified names (host.domain.xxx) rather than simple host names.

12.3.1 Error Messages from a Master Configuration: In a master/slave configuration, you can get the following error messages from a master. The error message is followed by a possible explanation:

12.3.1.1 Cannot resolve slave name XXX

To contact the slave, the slave name given in the configuration file must be resolved to an IP address. In this case, apcupsd could not get the IP address. Either the slave name is incorrect, your DNS may not be working, or you have started apcupsd during the boot process before the network is operational.

12.3.1.2 Got slave shutdown from SSS

This message should not be printed as it is not yet used.

12.3.1.3 Cannot write to slave SSS

This message occurs when the master attempts to send a message to the slave SSS and gets an error. It indicates that either the slave machine is not responding (apcupsd died, the system crashed, ...) or that the network is down.

12.3.1.4 Cannot read magic from slave SSS

This message indicates that the master attempted to read the code key from the slave SSS and it did not match the value expected. A common cause of this problem is that the master and slave versions of apcupsd are not the same. Please be sure you are running the same version of apcupsd on all your master and slave machines.

12.3.1.5 Connect to slave SSS failed

This message is logged when the master attempts to connect to slave SSS and no connection is accepted. The most common cause of this problem is that the slave copy of apcuspd is not yet ready to accept connections or is not running. Generally, apcupsd will retry the connection a bit later. If the problem is persistent, it can indicate a network problem or the slave name on the SLAVE directive of the master's configuration file is incorrect.

12.3.1.6 Cannot open stream socket

This indicates a fundamental networking problem on your system – either a lack of sufficient resources or you have not configured TCP/IP operations.

12.3.2 Error Messages from a Slave Configuration: In a master/slave configuration, you can get the following error messages from a slave. The error message is followed by a possible explanation:

12.3.2.1 Cannot resolve master name MMM

This message is logged when the slave attempts to resolve the name given on the MASTER configuration directive to an IP address. It probably means that the master name MMM is not defined, your DNS is not properly working, or you have started apcupsd in the boot process before the network is initialized. Check the name MMM, or use an explicit IP address on the MASTER configuration directive in the slave's configuration file.

12.3.2.2 Cannot bind local address, probably already in use

This means that the slave has attempted to bind the port number so that it can listen for messages from the master. This can occur if already have a copy of apcupsd running, or you have previously run apcupsd in the past 5 or 10 minutes, because occasionally the operating system will not shutdown a port correctly for 5 to 10 minutes after a program exits. In this case, you can either wait a few minutes for the problem to go away, or use a different port in both your master and slave configuration files.

12.3.2.3 Socket accept error

The slave got an error waiting on the accept() system call. This is probably due to a fundamental networking problem.

12.3.2.4 Unauthorized attempt from master MMM

The master named MMM (probably an IP address) contacted the slave but MMM is not the master that was listed on the MASTER configuration directive in /etc/apcupsd.conf, and consequently, it is not authorized to communicate with the slave. Please check that your MASTER and SLAVE names in your slave and master configuration files respectively are correct.

12.3.2.5 Read failure from socket

The slave got an error reading the socket open to the master. This indicates a fundamental networking problem.

12.3.2.6 Bad APC magic from master: MMM

The slave received a code key from the master that does not correspond to the one expected by the slave. The most common cause of this problem is that you are running a different version of apcupsd on the master and the slave. Please ensure that you are running the same version of apcupsd on all your master and slaves.

12.3.2.7 Bad user magic from master: MMM

This message indicates that the master and slave have previously communicated, but that the code key transmitted with the most recent message from the master does not correspond to what the slave expects. This problem is probably due to a network error or some other user or machine contacting the slave on the network port.

12.3.3 Master/Slave Connection Not Working: Master/slave problems are usually related to one of the following items:

1. Improper apcupsd.conf files. A good starting point are the master/slave example files in the examples subdirectory of the source.
2. Master or slave IP address or name incorrect. Try ping'ing each machine from the other using the names or addresses that you have put in the respective apcupsd.conf files.
3. Make sure no other program is using socket number 6666 or change the NETPORT directive in both apcupsd.conf files.

4. Make sure you are using the same version of apcupsd on both the master and slave machines.

13 Controlling Multiple UPSes on one Machine

You may want to use your server to control multiple UPSes. This is possible by proper configuration and by running one copy of apcupsd for each UPS to be controlled (recall the Configuration types.).

13.1 Configuration

The way to accomplish the above is to ensure that none of the critical files used by each of the two copies of apcupsd are the same. By using suitable configuration options, this is possible.

13.1.1 The First Copy of apcupsd: For example, assuming you have SmartUPSes in both cases, to configure and install the first copy of apcupsd, which controls a UPS and Computer A, one could use the following configuration:

```
./configure \
  --prefix=/usr \
  --sbindir=/sbin \
  --with-cgi-bin=/home/http/cgi-bin \
  --enable-cgi \
  --with-css-dir=/home/http/css \
  --with-log-dir=/etc/apcupsd \
  --with-serial-dev=/dev/ttyS0 \
  --enable-pthreads \
  --with-nis-port=3551 \
  --enable-powerflute
```

This is pretty much a "normal" installation using many of the defaults. Once built and installed, this would control the first UPS and cause a shutdown of the system when the batteries are low. This copy of apcupsd will be started and stopped automatically when the system is booted and halted.

13.1.2 The Second Copy of apcupsd: To configure and install the second copy of apcupsd, which controls the second UPS and Computer B, you could use the following configuration:

```
./configure \
  --prefix=$HOME/apcupsd/bin \
  --sbindir=$HOME/apcupsd/bin \
  --enable-cgi \
  --with-cgi-bin=$HOME/apcupsd/bin \
  --with-log-dir=$HOME/apcupsd/bin \
  --with-pid-dir=$HOME/apcupsd/bin \
  --sysconfdir=$HOME/apcupsd/bin \
  --with-lock-dir=$HOME/apcupsd/bin \
  --with-pwrfail-dir=$HOME/apcupsd/bin \
  --with-serial-dev=/dev/ttyS1 \
  --enable-pthreads \
  --with-nis-port=7001 \
  --disable-install-distdir
```

Note, in this case, we use considerably more configuration options to ensure that the system files are placed in a different directory ($HOME/apcupsd/bin). We have also selected a different serial port and a different NIS (Network Information Server) port. And finally, we have used the `--disable-install-distdir` option, which prevents `make install` from doing the final system installation (i.e. the modification of the halt script) since this was previously done.

13.1.3 Important Steps after Installation of the Second Copy: After the `make install` of the second copy of apcupsd there are a number important steps to complete. You must either remove or modify the file $HOME/apcupsd/bin/apccontrol, so that it will not shutdown Computer A when the battery of UPS 2 is low. One suggestion is to copy examples/safe.apccontrol into $HOME/apcupsd/bin/apccontrol. Alternatively, you could edit the $HOME/apcupsd/bin/apccontrol and delete all statements that attempt to shutdown the machine. Another important step is to find a way to shutdown Computer B when UPS 2's battery is low. Probably the simplest way to do this is to edit $HOME/apcupsd/bin/apcupsd.conf on Computer A so that this second copy of apcupsd becomes a network master. Then install a standard slave configuration on Computer B. Please remember that if UPS 1's batteries are exhausted before UPS 2's batteries, Computer B may not be properly shutdown. And at the current time, there is no simple means to make the two copies of apcupsd running on Computer A communicate. Thus there are certain risks in such a configuration. However, these configurations can be very useful for powering electronic equipment and such.

If Computer B is vitally important, it would probably be better to purchase a serial port card for it, or perhaps use a USB UPS. To ensure that it is

properly shutdown if Computer A goes down, you could run a second copy of apcupsd on Computer B as a slave connected to the main copy of apcupsd on Computer A. Thus Computer B would be running two slaves, one driven by the master controlling UPS 1 and the other by the master controlling UPS 2, and Computer B could be shutdown by the first master that signaled it to do so.

14 Support for SNMP UPSes

snmp To run apcupsd with an SNMP UPS, you need the following things:

- An SNMP UPS, for example a Web/SNMP card installed into the SmartSlot.
- apcupsd version 3.10.0 or higher
- Net-SNMP library (previously known as ucd-snmp) installed

14.1 Connecting an SNMP UPS

The Simple Network Management Protocol provides an interface to connect to remote devices through the network. apcupsd is now capable of using the SNMP interface of an SNMP-enabled UPS to communicate with an UPS. Currently apcupsd supports only APC's PowerNet MIB. To enable the SNMP support it is enough to configure the correct device in your apcupsd.conf configuration file. The directive needed for this configuration is:

```
DEVICE 192.168.100.2:161:APC:private
```

where the directive is made by four parts:

- IP address of the remote UPS
- Remote SNMP port
- Kind of remote SNMP agent, currently can only be "APC" for APC's powernet MIB
- The read-write community string, usually it is "private" for read-write access.

14.2 Building and Installing apcupsd

Follow the instructions in Building and Installing apcupsdl, being sure to include the following options (in addition to any others you need) on the **./configure** line:

```
./configure \
--with-serial-dev=<your-SNMP-device> \
--with-upstype=snmp \
--with-upscable=smart \
--enable-pthreads \
--enable-snmp
```

14.3 SNMP Specific Information

The SNMP connection gives less information compared to a serial smart cable. This is not a problem as the most useful information is given, together with a number of secondary parameters that are informative enough to run safely your UPS.

14.4 Known Problems

Currently (as of 3.10.0) the code to power off the UPS needs special configuration. The killpower command for SNMP UPSes can not be issued during shutdown as typically at some time during shutdown operations the network stack is stopped. To overcome this problem it is needed to modify the /etc/rc.d/apcupsd system control script to tell apcupsd to issue the power down command (killpower) to the UPS immediately before apcupsd initiates the system shutdown. For this reason it is paramount to set your UPS grace time to a value greater than 120 seconds to allow for clean shutdown operations before the UPS removes the power from its plugs. To enable correct shutdown operation during powerdown do the following:

- Connect to your Web/SNMP card using your favorite web browser, go to the UPS configuration menu and change the "Shutdown Delay" parameter to 180 seconds or more, depending on how much time your system shutdown requires to umount all the filesystems.
- Change /etc/rc.d/apcupsd script adding the '—kill-on-powerfail' to the apcupsd invocation.

- Restart your apcupsd

With this setup your UPS operations should be safe.

15 Alternate Ways To Run The Network Information Server

apcupsd maintains STATUS and EVENTS data concerning the UPS and its operation. This information can be obtained over the network using either apcnisd or apcupsd's internal network information server, which is essentially the same code as apcnisd but compiled into apcupsd. Clients on the network make a connection to the information server and send requests for status or events data, which the server then transmits to them.

The information served to the network by this interface should not be confused with master/slave mode that shares a UPS between two or more computers. That code is described in Configuration Directives for Sharing a UPS of this documentation.

There are three different ways to run the information server depending on your requirements and preferences. It can be run as 1. a standalone program, 2. a standalone program invoked by the inetd daemon, or 3. as a thread (or child process) of apcupsd (default configuration). We recommend option 3 unless you have specific reasons to do otherwise. Option 3 is what is configured in by default.

15.1 Running the server as a child of apcupsd

This is probably the simplest way to run the network information server. To do so, you simply make sure the **NETSERVER** directive in /etc/apcupsd/apcupsd.conf is **on**, and then stop and restart apcupsd. It will automatically create the server thread (or spawn an additional child process named apcnis) to handle network clients. In the case where pthreads are enabled, a new thread will be created rather than a child process to handle the network information requests. Note, the above modification should not be necessary if you use the default apcupsd.conf, since it is already turned on.

Although this method is simple, it affords no protection from the outside world accessing your network server unless you are behind a firewall. In addition, if there is a bug in the network server code, or if a malicious user sends bad data, it may be possible for apcnis to die, in which case, though it is not supposed to, apcupsd may also exit, thus leaving your machine

without shutdown protection. In addition, since apcupsd is running at root level, all threads or any child process will do so also. That being said, most of us prefer to run the server this way.

With apcupsd version 3.8.2 and later, you may enable the TCP Libwrap subroutines to add additional security. In this case, access to the network server will be controlled by the statements you put in /etc/hosts.allow.

15.2 Running apcnisd from INETD

This is probably the most secure and most desirable way of running the network information server. Unfortunately, it is a bit more complicated to set up. However, once running, the server remains unexecuted until a connection is attempted, at which point, inetd will invoke apcnisd. Once apcnisd has responded to the client's requests, it will exit. None of the disadvantages of running it standalone apply since apcnisd runs only when a client is requesting data. Note, running in this manner works only if you are using the old forking code and have pthreads explicitly turned off. The pthreads version of apcupsd does not support the shared memory calls that are necessary for apcnisd to access the internal state of apcupsd.

An additional advantage of this method of running the network information server is that you can call it with a TCP wrapper and thus use access control lists (ACL) such as hosts.allow. See the man pages for hosts.allow for more details.

To configure apcnisd to run from INETD, you must first put an entry in /etc/services as follows:

```
apcnisd         3551/tcp
```

This defines the port number (3551) and the service (TCP) that apcnisd will be using. This statement can go anywhere in the services file. Normally, one adds local changes such as these to the end of the file.

Next, you must modify /etc/inetd.conf to have the following line:

```
apcnisd stream  tcp     nowait  root    /usr/sbin/tcpd  /sbin/apcnisd -i
```

If you do not want to run the TCP wrapper, then the line should be entered as follows (not tested):

```
apcnisd stream  tcp     nowait  root    /sbin/apcnisd -i
```

Please check that the file locations are correct for your system. Also, note that the `-i` option is necessary so that apcnisd knows that it was called by INETD. Before restarting INETD, first ensure that the **NETSERVER** directive in /etc/apcupsd/apcupsd.conf is set to **off**. This is necessary to prevent apcupsd from starting a child process that acts as a server. If you change **NETSERVER**, you must stop and restart apcupsd for the configuration change to be effective.

Finally, you must restart INETD for it to listen on port 3551. On a Red Hat system, you can do so by:

```
/etc/rc.d/init.d/inet reload
```

At this point, when a client attempts to make a connection on port 3551, INETD will automatically invoke apcnisd.

15.3 Running apcnisd Standalome

This is probably the least desirable of the three ways to run an apcupsd network information server because if apcupsd is stopped, you must also stop apcnisd before you can restart apcupsd. This is because apcnisd, when run standalone, holds the shared memory buffer by which apcnisd and apcupsd communicate. This prevents a new execution of apcupsd from creating it.

To execute apcnisd in standalone mode, first ensure that the **NETSERVER** directive in **/etc/apcupsd/apcupsd.conf** is set to **off**. This is necessary to prevent apcupsd from starting a child process that acts as a server. Restart apcupsd normally, then:

```
/sbin/apcnisd
```

The advantage of running the network information server standalone is that if for some reason, a client causes the network server to crash, it will not affect the operation of apcupsd.

16 apcupsd System Logging

The apcupsd philosophy is that all logging should be done through the **syslog** facility (see: `man syslog`). This is now implemented with the exceptions that **STATUS** logging, for compatibility, with prior versions is still done to a file, and **EVENTS** logging can be directed to a "temporary" file so that it can be reported by the network information server.

16.1 Logging Types

apcupsd splits its logging into four separate types called:

1. DEBUG
2. DATA
3. STATUS
4. EVENTS

Debug logging consists of debug messages. Normally these are turned on only by developers, and currently there exist very few of these debug messages.

Logging

Data logging consists of periodically logging important data concerning the operation of the UPS. See the Data Logging (see DATA Logging) section of this manual for more details.

Logging

Status logging consists of logging all available information known about your UPS as a series of ASCII records. This information is also made available by the apcupsd network information server.

For more details on STATUS logging, see the Status (see apcupsd Status Logging) section of the Technical Reference.

Logging

Events logging consists of logging events as they happen. For example, successful startup, power fail, battery failure, system shutdown, ...

See the manual section on customizing event handling (see Customizing Event Handling) for more details.

16.2 Implementation Details

In order to ensure that the data logged to syslog() can be directed to different files, I have assigned syslog() levels to each of our four types of data as follows:

1. 1. DEBUG logging has level LOG_DEBUG
2. 2. DATA logging has level LOG_INFO
3. 3. STATUS logging has level LOG_NOTICE
4. 4. EVENTS logging has levels LOG_WARNING, LOG_ERR, LOG_CRIT, and LOG_ALERT

It should be noted that more work needs to be done on the precise definitions of each of the levels for EVENTS logging. Currently, it is roughly broken down as follows:

LOG_WARNING general information such as startup, etc.

LOG_ERR an error condition detected, e.g. communications problem with the UPS.

LOG_CRIT a serious problem has occurred such as power failure, running on UPS batteries, ...

LOG_ALERT a condition that needs immediate attention such as pending system shutdown, ...

The default Facility for syslog() logging is DAEMON, although this can be changed with the FACILITY directive in apcupsd.conf. In the following example, we should the facility as local0.

More work needs to be done to the code to ensure that it corresponds to the above levels.

As a practical example of how to setup your syslog() to use the new logging feature, suppose you wish to direct all DATA logging to a file named /var/log/apcupsd.data, all EVENTS to the standard /var/log/messages file (to be mixed with other system messages), and at the same time send all

EVENTS to /var/log/apcupsd.events, and finally, you want to send all STATUS logging to the named pipe /var/log/apcupsd.status

First as root, you create the named pipe:

```
mkfifo /var/log/apcupsd.status
```

Change its permissions as necessary or use the -m option to set them when creating the pipe.

Then you modify your /etc/syslog.conf file to direct the appropriate levels of messages where you want them. To accomplish the above, my syslog.conf file looks like:

```
# exclude all apcupsd info by default
*.info;local0.none                    /var/log/messages

# Everything for apcupsd goes here
local0.info;local0.!notice             /var/log/apcupsd.data
local0.notice;local0.!warn            |/var/log/apcupsd.status
local0.warn                            /var/log/apcupsd.events
local0.warn                            /var/log/messages
```

16.3 Developers Notes

All logging functions and all error reporting are now done through the log_event() subroutine call. Exceptions to this are: initialization code where printf's are done, and writing to the status file. Once the initialization code has completed and the fork() to become a daemon is done, no printf's are used. log_event() has exactly the same format as syslog(). In fact, the subroutine consists of only a syslog() call. If anyone really wishes to log to a file, the code to do so can easily be done by adding code to log_event() in apclog.c.

Installation: Windows

17 The Windows Version of apcupsd

The Windows version of apcupsd has been tested on Win95, Win98, WinMe, WinNT, WinXP, and Win2000 systems. This version of apcupsd has been

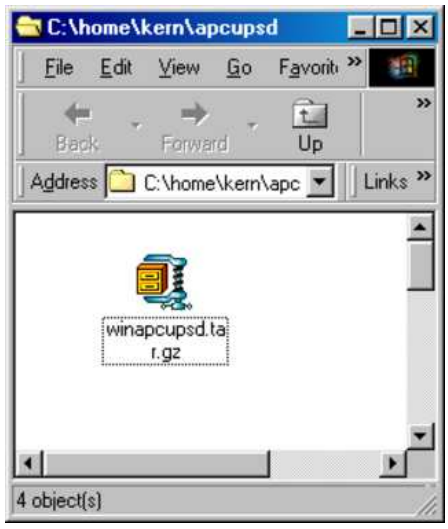
built to run under the CYGWIN environment, which provides many of the features of Unix on Windows systems. It also permitted a rapid port with very few source code changes, which means that the Windows version is for the most part running code that has long proved stable on Unix systems. Even though the Win32 version of apcupsd is a port that relies on many Unix features, it is just the same a true Windows program. When running, it is perfectly integrated with Windows and displays its icon in the system icon tray, and provides a system tray menu to obtain additional information on how apcupsd is running (status and events dialogue boxes). If so desired, it can also be stopped by using the system tray menu, though this should normally never be necessary.

Once installed apcupsd normally runs as a system service. This means that it is immediately started by the operating system when the system is booted, and runs in the background even if there is no user logged into the system.

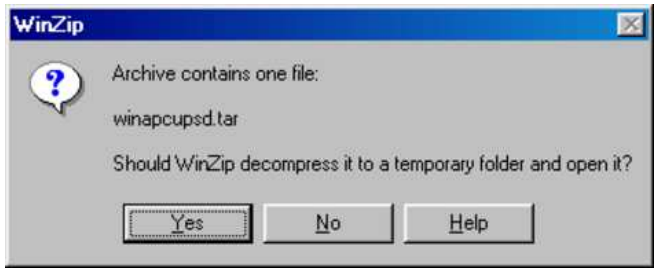
17.1 Installation

Normally, you will install the Windows version of apcupsd from the binaries. This install is somewhat Unix like since you do many parts of the installation by hand. To install the binaries, you need **WinZip**.

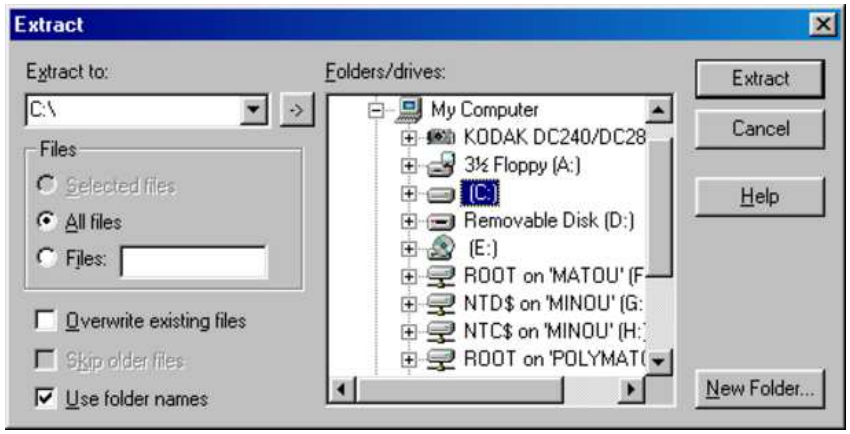
- Simply double click on the winapcupsd-3.8.5.tar.gz icon. The actual name of the icon will vary from one release version to another.



- When Zip says that it has one file and asks if it should unpack it into a temporary file, respond with **Yes**.



- Ensure that you extract all files and that the extraction will go into C:\



If you wish to install the package elsewhere, please note that you will need to proceed with a manual installation, which is not particularly easy as you must rebuild the source and change the configuration file as well.

This installation assumes that you do **not** have CYGWIN installed on your computer. If you do, and you use mount points, you may need to do a special manual installation.

Once you have unzipped the binaries, open a window pointing to the binary installation folder (normally c:\apcupsd). This folder should contain folders with the name bin, etc, examples, and manual. If and when you no longer need them, the examples and manual sub-folders of the c:\apcupsd directory may be removed.

Continuing the installation process:

- Open the directory c:\apcupsd\etc\apcupsd in the Windows Explorer by Clicking on the apcupsd folder then on the etc folder, then on the apcupsd folder. Finally double click on the file apcupsd.conf and edit it to contain the values appropriate for your site. In most cases, no changes will be needed, but if you are not using COM1 for your serial port, you will need to set the **DEVICE** configuration directive to the correct serial port. Note, if you are using WinNT or Win2000, the operating system may probe the port attempting to attach a serial mouse. This will cause apcupsd to be unable to communicate with the serial port. If this happens, or out of precaution, you can edit the c:\boot.ini file. Find the line that looks something like the following:

  multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 4.00"

  and add the following to the end of the line: /NoSerialMice:COM1 (or COM2 depending on what you want to use). The new line should look similar to:

  multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 4.00" /NoSerialMice:COM1

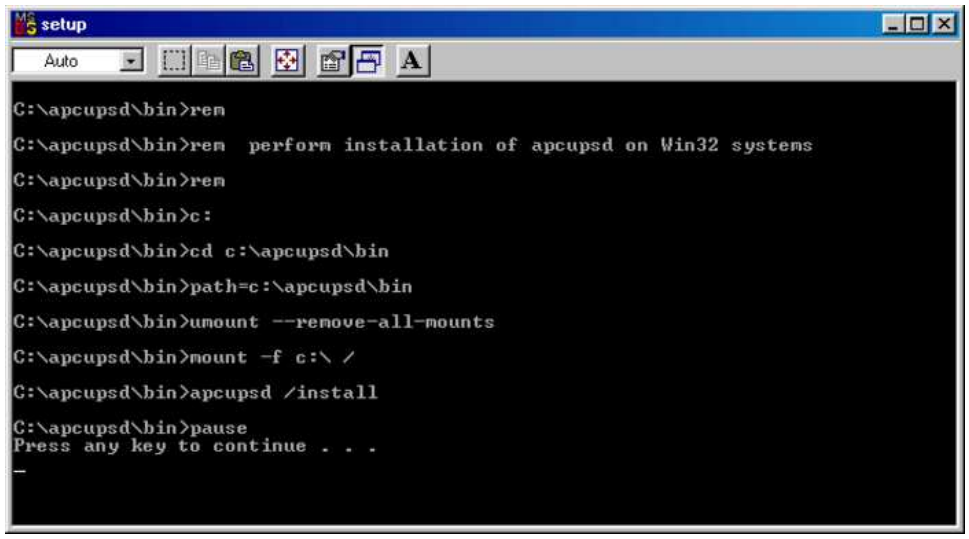
  where the only thing you have changed is to append to the end of the line. This addition will prevent the operating system from interferring with apcupsd

- Then return to c:\apcupsd and open on the bin folder so that you see its contents.

- To do the final step of installation, double click on the setup.bat program. This script will setup the appropriate mount points for the

directories that apcupsd uses, it will install apcupsd in the system registry, and on Windows 98, it will start apcupsd running.

If everything went well, you will get something similar to the following output in a DOS shell window:



What is important to verify in the DOS window is that the root directory \ is mounted on device c:\.

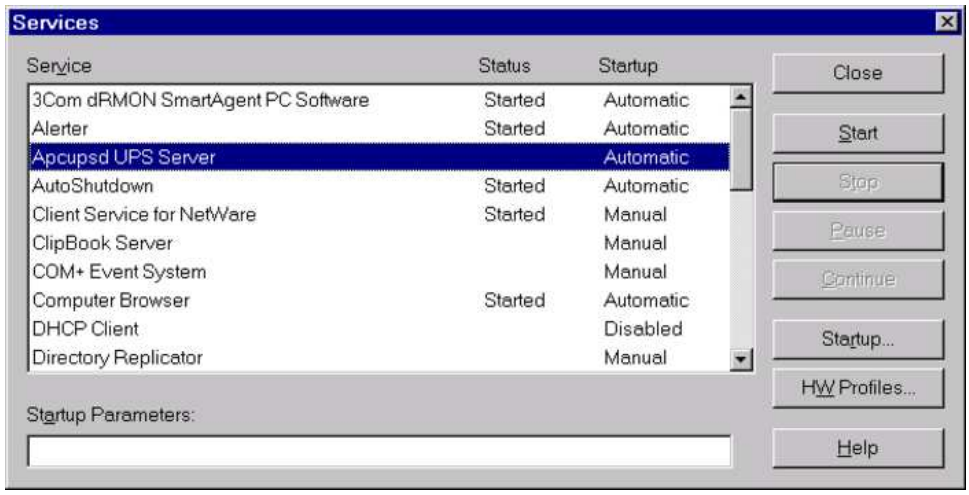
The DOS window will be followed immediately by a Windows dialogue box as follows:

- On Windows 98, to actually start the service, either reboot the machine, which is not necessary, or open a DOS shell window, and type the following commands:

```
cd c:\apcupsd\bin
apcupsd /service
```

  Alternatively, you can go to the c:\apcupsd\bin folder with the Explorer and double click on the **Start** icon.

- On Windows NT, to start the service, either reboot the machine, which is not necessary, or go to the Control Panel, open the Services folder and start the apcupsd daemon program by selecting the apcupsd UPS Server and then clicking on the Start button as shown below:



  If the Services dialog reports a problem, it is normally because your DEVICE statement does not contain the correct serial port name.

You probably should also click on the **Startup...** button to ensure that the correct defaults are set. The dialogue box that appears should have **Startup Type** set to **Automatic** and **Logon** should be set to **System Account**

with **Allow Service to Interact with Desktop** checked. If these values are not set correctly by default, please change them otherwise apcupsd will not work.

For WinXP systems (and probably Win2K), the dialogs are a bit different from those shown here for WinNT, but he concept is the same. You get to the Services dialog by clicking on: **Control Panel -> Administrative Tools -> Component Services**. The apcupsd service should appear in the right hand window when you click on **Services (Local)** in the left hand menu window.

That should complete the installation process. When the system tray icon turns from a battery into a plug, right click on it and a menu will appear. Select the **Events** item, and the Events dialogue box should appear. There should be no error messages. By right clicking again on the system tray plug and selecting the **Status** item, you can verify that all the values for your UPS are correct.

When the UPS switches to the battery, the battery icon will reappear in the system tray. While the UPS is online, if the battery is not at least 99% charged, the plug icon will become a plug with a lightning bolt in the middle to indicate that the battery is charging.

17.2 Installation Directory

The Win32 version of apcupsd must reside in the c:\apcupsd\ directory, and there must be a c:\tmp directory on your machine. The installation will do this automatically, and we recommend that you do not attempt to place apcupsd in another directory. If you do so, you are on your own, and you will need to do a rebuild of the source.

17.3 Testing

It would be hard to overemphasize the need to do a full testing of your installation of apcupsd as there are a number of reasons why it may not behave properly in a real power failure situation.

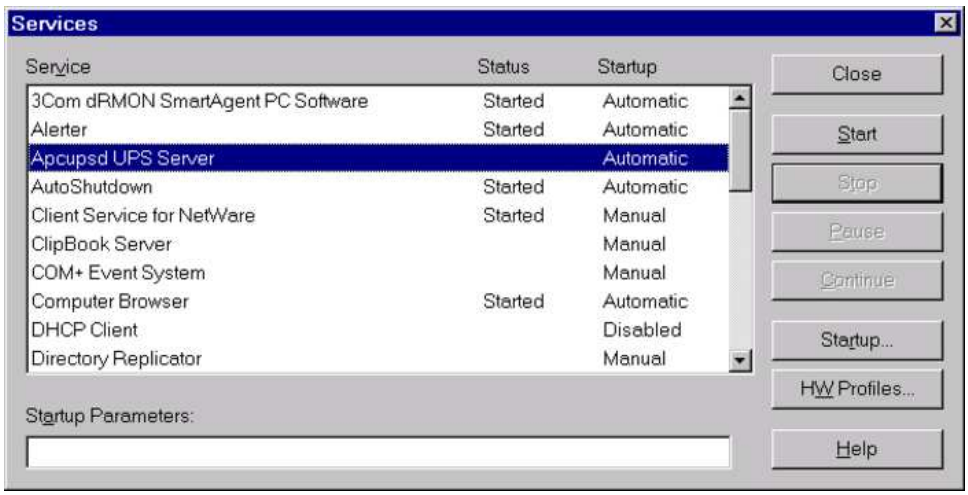
Please read Testing Apcupsd of this document for general instructions on testing the Win32 version. However, on Win32 systems, there is no Unix system log file, so if something goes wrong, look in the file c:\apcupsd\etc\apcupsd\apcupsd.events where apcupsd normally logs its events, and you will generally find more detailed information on why the

program is not working. The most common cause of problems is either improper configuration of the cable type, or an incorrect address for the serial port.

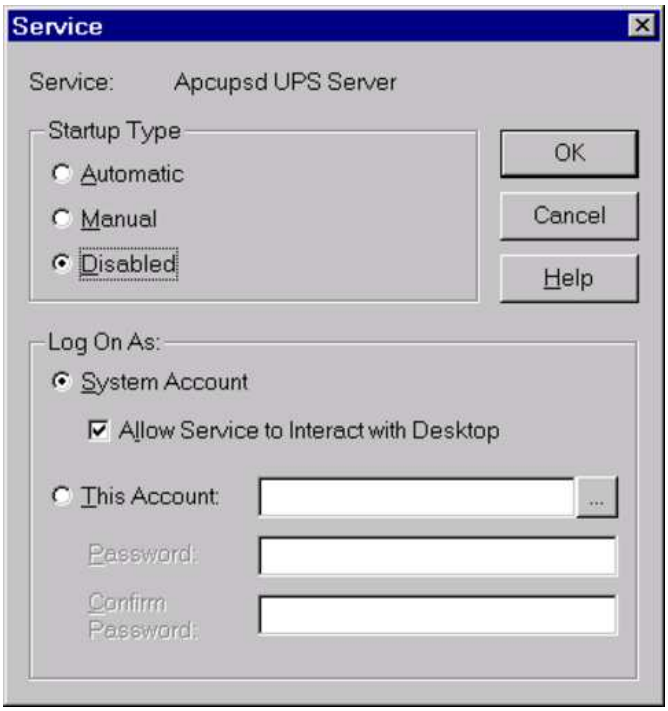
17.4 Upgrading

On Win98 and Win95 systems, to upgrade to a new release, simply stop apcupsd by using the tray icon and selecting the **Close apcupsd** menu item, or by double clicking on the **Stop** icon located in the c:\apcupsd\bin directory, then apply the upgrade and restart apcupsd.

On WinNT systems (and Win2000 systems), you may stop apcupsd as indicated above or alternatively you may stop apcupsd by using the **Services** item in the **Control Panel**. In addition, at least on my system, there seems to be a WinNT bug that causes the system to prevent **apcupsd.exe** from being overwritten even though the file is no longer being used. This is manifested by an error message when attempting load a new version and overwrite the old apcupsd.exe (the extract part of WinZip as described above). To circumvent this problem (if it happens to you), after shutting down the running version of apcupsd, through the **Services** dialogue in the **Control Panel**, first click on the **Stop button**:



then click on the **Startup ...** button, and in the Startup dialogue select the **Disabled** button to disable apcupsd:



After closing the dialogues, reboot the system, typical of Microsoft :-(. When the system comes back up, apcupsd will not be automatically launched as a service, and you can install the new version. To reinstate apcupsd as an automatic service, using the **Control Panel**: reset apcupsd to **Automatic** startup in the Startup dialogue, then restart apcupsd in the **Services** dialogue as shown above in the installation instructions. Frequently after an upgrade, you will click on the **Start** button and after a few seconds, the system reports that it failed to start. The cause of this problem is unknown, but the solution is simply to click again on the **Start** button.

17.5 Post Installation

After installing apcupsd and before running it, you should check the contents of two files to ensure that it is configured properly for your system. The first is c:\apcupsd\etc\apcupsd\apcupsd.conf. You will probably need to change your UPSCABLE directive, your UPSTYPE and possibly your DEVICE directives. Please refer to the configuration section of this manual for more details.

The second file that you should examine is c:\apcupsd\etc\apcupsd\apccontrol. This file is called by apcupsd when events (power loss, etc) are generated. It permits the user to program

handling the event. In particular, it permits the user to be notified of the events. For the Win32 version, each event is programmed to display a Windows popup dialogue box. If your machine is mostly unattended, you may want to comment out some of these popup dialogue boxes by putting a pound sign (#) in column one of the appropriate line.

17.6 Problem Areas

In addition to possible problems of reinstallation or upgrade on WinNT systems, as noted above, we have discovered the following problem: On some Windows systems, the domain resolution does not seem to work if you have not configured a DNS server in the Network section of the Control Panel. This problem should be apparent only when running a master or a slave configuration. In this case, when you specify the name of the master or the slave machine(s) in your apcupsd.conf file, apcupsd will be unable to resolve the name to a valid IP address. To circumvent this problem, simply enter all machine addresses as an IP address rather than a domain name, or alternatively, ensure that you have a valid DNS server configured on your system (often not the case on Win32 systems). For example, instead of using the directive "MASTER my.master.com" use something like "MASTER 192.168.1.54" where you replace the IP address with your actual IP address.

Also, on WinNT systems, the PIF files in /apcupsd/bin used for starting and stopping apcupsd do not work. Use the services control panel instead.

On Win95 systems, there are reports that the PIF files do not work. If you find that to be the case, the simplest solution is to use the batch files that we have supplied in the c:/apcupsd/bin directory. Also, on Win95 systems, we have an unconfirmed report that indicates that apcupsd does not start automatically as a service even though the Registry has been properly updated. If you experience this problem, a work around is to put a shortcut to apcupsd in the StartUp folder.

As noted above, after an upgrade, you may need to start apcupsd several times before it will actually run.

On WinNT, WinXP, and Win2K systems, you can examine the System Applications log to which apcupsd writes Windows error messages during startup.

Regardless of which Windows system you are running, apcupsd logs most error messages to c:\apcupsd\etc\apcupsd\apcupsd.events. This type error messages such as configuration file not found, etc are written to this file.

17.7 Utility Functions

The directory c:\apcupsd\bin contains six utility routines (actually .pif files) that you may find useful. They are:

```
Start
Stop
Install
Uninstall
ups-events
ups-status
```

Any of these utilities may be used on any system, with the exception of the Start utility, which cannot be used on WinNT and Win2000 systems. On those systems, the apcupsd service must always be started through the **Services** sub-dialogue of the **Control Panel**.

The **Install** and **Uninstall** utilities install and uninstall apcupsd from the system registry only. All other pieces (files) of apcupsd remain intact. It is not absolutely necessary for apcupsd to be installed in the registry as it can run as a regular program. However, if it is not installed in the registry, it cannot be run as a service.

The functions of **Stop**, **ups-events**, and **ups-status** can be more easily invoked by right clicking on the apcupsd icon in the system tray and selecting the desired function from the popup menu.

17.8 Disclaimer

Some of the features such as EEPROM programming have not been exhaustively tested on Win32 systems. If at all possible, we recommend not to use it as a network master on Win95, Win98, and WinMe due to the instability of those operating systems.

Some items to note:

- This version of apcupsd will not attempt to shut off the UPS power when the battery is exhausted. Thus if the power returns before the UPS completely shuts down, your computer may not reboot automatically. This is because we do not know how to regain control after the disks have been synced in order to shut off the UPS power.

Nevertheless, it is possible to use the `--kill-on-powerfail` option on the apcupsd command line, but the use of this option could cause the power to be cut off while your machine is still running. See Shutdown Sequence of this document for a more complete discussion of this subject. If you are still interested in trying to get this to work, please look at the code that is commented out in c:\apcupsd\etc\apcupsd\apccontrol under the **doshutdown** case.

An alternative to the `--kill-on-powerfail` option is to use the KILLDELAY (see KILLDELAY <time in seconds>) configuration directive.

This configuration directive is appropriate on Windows machines where apcupsd continues to run even when the machine is halted (as is the case on most NT machines).

- When apcupsd detects important events, it calls c:\apcupsd\etc\apcupsd\apccontrol, which is a Unix shell script. You may modify this script to suit your particular needs. Currently, it puts a Windows dialogue on the screen with a brief explanation of the event. If these dialogues annoy you, you can remove or comment out the calls to **popup** from this file.

17.9 Email Notification of Events

On Win95/98 systems, it is possible to receive notification of apcupsd events that are passed to apccontrol. This is possible using a simple email program that unfortunately is not functioning 100% correctly. In addition, I (Kern) was not able to make this program work on WinNT while apcupsd is running as a service under the system account (it works fine with any user account).

If you wish to try this program on Win95/98 systems, look at the files named **changeme, commfailure, commok, onbattery, and mainsback** in the directory c:\apcupsd\examples. To use them, you must modify the SYSADMIN variable to have a valid email address, then copy the files into the directory c:\apcupsd\etc\apcupsd.

17.10 Killpower under Windows

If your batteries become exhausted during a power failure and you want your machine to automatically reboot when the power comes back, it is useful to implement the killpower feature of the UPS where apcupsd sends the UPS the command to shut off the power. In doing so, the power will be cut to

your PC and if your BIOS is properly setup, the machine will automatically reboot when the power comes back. This is important for servers.

This feature is implemented on Unix systems by first requesting a system shutdown. As a part of the shutdown, apcupsd is terminated by the system, but the shutdown process executes a script where apcupsd is recalled after the disks are synced and the machine is idle. Apcupsd then requests the UPS to shut off the power (killpower).

Unfortunately on Windows, there is no such shutdown script that we are aware of and no way for apcupsd to get control after the machine is idled. If this feature is important to you, it is possible to do it by telling apcupsd to immediately issue the killpower command after issuing the shutdown request. The danger in doing so is that if the machine is not sufficiently idled when the killpower takes place, the disks will need to be rescanned (and there is a possibility of lost data however small). Generally, UPSes have a shutdown grace period which gives sufficient time for the OS to shutdown before the power is cut.

To implement this feature, you need to add the `-p` option to the apcupsd command line that is executed by the system. Currently the procedure is manual. You do so by editing the registry and changing the line:

```
c:\apcupsd\apcupsd.exe /service
```

found under the key:

```
HKEY_LOCAL_MACHINE Software\Microsoft\Windows\CurrentVersion\RunServices
```

to

```
c:\apcupsd\apcupsd.exe /service -p
```

If you have a Smart UPS, you can configure the kill power grace period, and you might want to set it to 3 minutes. If you have a dumb UPS, there is no grace period and you should not use this procedure. If you have a Back-UPS CS or ES, these UPSes generally have a fixed grace period of 2 minutes, which is probably sufficient.

17.11 Power Down During Shutdown

Our philosophy is to shutdown a computer but not to power it down itself (as opposed to having the UPS cut the power as described above). That is we prefer to idle a computer but leave it running. This has the advantage that in a power fail situation, if the killpower function described above does not work, the computer will continue to draw down the batteries and the UPS will hopefully shutoff before the power is restore thus permitting an automatic reboot.

Nevertheless some people prefer to do a full power down. To do so, you might want to get a copy of PsShutdown, which does have a power down option. You can find it and a lot more useful software at: http://www.sysinternals.com/ntw2k/freeware/pstools.shtml. to use their shutdown program rather than the apcupsd supplied version, you simply edit:

```
c:\apcupsd\etc\apcupsd\apccontrol
```

with any text editor and change our calls to shutdown to psshutdown.

17.12 Command Line Options Specific to the Windows Version

These options are not normally seen or used by the user, and are documented here only for information purposes. At the current time, to change the default options, you must either manually run apcupsd or you must manually edit the system registry and modify the appropriate entries.

In order to avoid option clashes between the options necessary for apcupsd to run on Windows and the standard apcupsd options, all Windows specific options are signaled with a forward slash character (/), while as usual, the standard apcupsd options are signaled with a minus (-), or a minus minus (—). All the standard apcupsd options can be used on the Windows version. In addition, the following Windows only options are implemented:

/servicehelper Run the service helper application

/service Start apcupsdas a service

/run Run the apcupsd application

/install Install apcupsd as a service in the system registry

/remove Uninstall apcupsd from the system registry

/about Show the apcupsd about dialogue box

/status Show the apcupsd status dialogue box

/events Show the apcupsd events dialogue box

/kill Stop any running apcupsd

/help Show the apcupsd help dialogue box

It is important to note that under normal circumstances the user should never need to use these options as they are normally handled by the system automatically once apcupsd is installed. However, you may note these options in some of the .pif files that have been created for your use.

17.13 Building the Win32 Version from the Source

If you have the source code, follow the standard procedures for building apcupsd on Unix in Building and Installing apcupsd of this manual. Please don't forget to look at the system specifics for CYGWIN.

Installation: Serial-Line UPSes

18 Overview of Serial-Interface UPSes

If you have a UPS that communicates via serial port, you need to do two things before you can even think about configuring the software. First, you need to figure out whether it's a dumb (voltage-signalling) UPS or speaks the apcsmart protocol (see this discussion (see upstypes)). Second, if you have an interface cable from APC, you need to figure out what kind it is. If you don't have such a cable, you need to build one. A straight-through serial cable won't work (see crazy).

According to Bill Marr the Belkin F5U109, also sold as F5U409 also works with apcupsd for kernel versions 2.4.25 or higher and kernels 2.6.1 and higher. These newer kernels are needed to have the patch that makes the mct_u232 (Magic Control Technology) module and other adapters work with RS-232 devices that do not assert the CTS signal.

19 Connecting a Serial-Line UPS to a USB Port

By using a special adaptor, you can connect your serial-line UPS to a USB port. If you would like to free up your serial port and connect your existing serial port UPS to a USB port, it is possible if you have one of the later kernels. You simply get a serial to USB adapter that is supported by the kernel, plug it in and make one minor change to your apcupsd.conf file and away you go. (Kern adds: Thanks to Joe Acosta for pointing this out to me.)

The device that Joe Acosta and Kern are using is IOgear GUC232A USB 2 serial adapter. Bill Marr informs us that it also works with a Back-UPS Pro 650 and the 940-0095B cable.

At Kern's site, running Red Hat 7.1 with kernel 2.4.9-12, he simply changed his /etc/apcupsd/apcupsd.conf configuration line to be:

```
DEVICE /dev/ttyUSB0
```

Depending on whether or not you have **hotplug** working, you may need to explicitly load the kernel modules **usbserial** and **pl2303**. In Kern's case, this was not necessary.

20 Connecting a APC USB UPS to either a PC USB or Serial Port

An interesting fact is that the USB ports (actually an RJ45 connector) on APC UPSes not only speak USB, but also serial apcsmart and dumb voltage-signalling as well! This is something that one of our users discovered by accident. With the Custom RJ45 cable (actually a RJ45 to serial cable) described below plugged into the APC UPS USB socket at one end and the other end plugged into a serial port on your PC, any APC USB UPS (except some of the newer really low cost models) will act as a serial-line device.

21 Cables

You can either use the cable that came with your UPS (the easiest if we support it) or you can make your own cable. We recommend that you obtain a supported cable directly from APC.

If you already have an APC cable, you can determine what kind it is by examining the flat sides of the two connectors where you will find the cable number embossed into the plastic. It is generally on one side of the male connector.

To make your own cable you must first know whether you have a UPS that speaks the apcsmart protocol or a "dumb" UPS that uses serial port line voltage signalling.

If you have an apcmart UPS, and you build your own cable, build a *Smart-Custom* cable. If you have a voltage-Signalling or dumb UPS, build a *Simple-Custom* cable. If you have a BackUPS CS with a RJ45 connector, you can build your own *Custom-RJ45* cable.

21.1 Smart-Custom Cable for SmartUPSes

```
  SMART-CUSTOM CABLE

Signal Computer                 UPS
       DB9F                    DB9M
 RxD    2   --------------------  2  TxD  Send
 TxD    3   --------------------  1  RxD  Receive
 GND    5   --------------------  9  Ground
```

When using this cable with apcupsd specify the following in apcupsd.conf:

If you have an OS that requires DCD or RTS to be set before you can receive input, you might try building the standard APC Smart 940-0024C cable listed below.

```
UPSCABLE smart
UPSTYPE apcsmart
DEVICE /dev/ttyS0 (or whatever your serial port is)
```

If you wish to build the standard cable furnished by APC (940-0024C), use the following diagram.

```
  APC Smart Cable 940-0024C

Signal Computer                 UPS
       DB9F                    DB9M
 RxD    2   --------------------  2  TxD  Send
```

```
TxD    3   --------------------  1  RxD  Receive
DCD    1   --*
             |
DTR    4   --*
GND    5   --------------------  9  Ground
RTS    7   --*
             |
CTS    8   --*
```

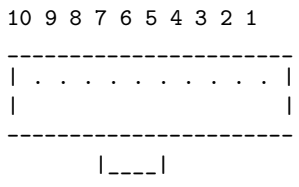
21.2 Smart Signalling Cable for BackUPS CS Models

If you have a BackUPS CS, you are probably either using it with the USB cable that is supplied or with the 940-0128A supplied by APC, which permits running the UPS in dumb mode. By building your own cable, you can now run the BackUPS CS models (and perhaps also the ES models) using smart signalling and have all the same information that is available as running it in USB mode.

The jack in the UPS is actually a 10 pin RJ45. However, you can just as easily use a 8 pin RJ45 connector, which is more standard (ethernet TX, and ISDN connector). It is easy to construct the cable by cutting off one end of a standard RJ45-8 ethernet cable and wiring the other end (three wires) into a standard DB9F female serial port connector.

Below, you will find a diagram for the CUSTOM-RJ45 cable:

```
  CUSTOM-RJ45 CABLE

Signal Computer              UPS     UPS
       DB9F                RJ45-8  RJ45-10
 RxD    2   ----------------  1      2     TxD  Send
 TxD    3   ----------------  7      8     RxD  Receive
 GND    5   ----------------  6      7     Ground
 FG  Shield ----------------  3      4     Frame Ground

The RJ45-8 pins are: looking at the end of the connector:

 8 7 6 5 4 3 2 1
-------------------
| . . . . . . . . |
|                 |
-------------------
       |____|

The RJ45-10  pins are: looking at the end of the connector:
```



For the serial port DB9F connector, the pin numbers are stamped in the plastic near each pin. In addition, there is a diagram near the end of this chapter.

Note, one user, Martin, has found that if the shield is not connected to the Frame Ground in the above diagram (not in our original schematic), the UPS (a BackUPS CS 500 EI) will be unstable and likely to rapidly switch from power to batteries (i.e. chatter).

When using this cable with apcupsd specify the following in apcupsd.conf:

```
UPSCABLE smart
UPSTYPE apcsmart
DEVICE /dev/ttyS0 (or whatever your serial port is)
```

The information for constructing this cable was discovered and transmitted to us by slither_man. Many thanks!

21.3 Voltage-Signalling Cable for "dumb" UPSes

NOTE. YOU DO NOT HAVE THIS CABLE UNLESS YOU BUILT IT YOURSELF. THE SIMPLE-CUSTOM CABLE IS NOT AN APC PRODUCT.

For "dumb" UPSes using voltage signalling, if you are going to build your own cable, we recommend to make the cable designed by the apcupsd team as follows:

```
        SIMPLE-CUSTOM CABLE

Signal Computer                  UPS
       DB9F   4.7K ohm          DB9M
 DTR    4   --[####]--*              DTR set to +5V by Apcupsd
                       |
```

```
CTS    8   ----------*---------  5  Low Battery
GND    5   --------------------  4  Ground
DCD    1   --------------------  2  On Battery
RTS    7   --------------------  1  Kill UPS Power
```

List of components one needs to make the Simple cable:

1. One (1) male DB9 connector, use solder type connector only.
2. One (1) female DB9/25F connector, use solder type connector only.
3. One (1) 4.7K ohm 1/4 watt 5% resistor.
4. resin core solder.
5. three (3) to five (5) feet of 22AWG multi-stranded four or more conductor cable.

1. Solder the resistor into pin 4 of the female DB9 connector.
2. Next bend the resistor so that it connects to pin 8 of the female DB9 connector.
3. Pin 8 on the female connector is also wired to pin 5 on the male DB9 connector. Solder both ends.
4. Solder the other pins, pin 5 on the female DB9 to pin 4 on the male connector; pin 1 on the female connector to pin 2 on the male connector; and pin 7 on the female connector to pin 1 on the male connector.
5. Double check your work.

We use the DTR (pin 4 on the female connector) as our +5 volts power for the circuit. It is used as the Vcc pull-up voltage for testing the outputs on any "UPS by APC" in voltage-signalling mode. This cable may not work on a BackUPS Pro if the default communications are in apcsmart mode. This cable is also valid for "ShareUPS" BASIC Port mode and is also reported to work on SmartUPSes. However, the Smart Cable described above is much simpler. To have a better idea of what is going on inside apcupsd, for the SIMPLE cable apcupsd reads three signals and sets three:

```
Reads:
CD, which apcupsd uses for the On Battery signal when high.
```

```
CTS, which apcupsd uses for the Battery Low signal when high.

RxD (SR), which apcupsd uses for the Line Down
    signal when high. This signal isn't used for much.

Sets:
DTR, which apcupsd sets when it detects a power failure (generally
     5 to 10 seconds after the CD signal goes high). It
     clears this signal if the CD signal subsequently goes low
     -- i.e. power is restored.

TxD (ST), which apcupsd clears when it detects that the CD signal
     has gone low after having gone high - i.e. power is restored.

RTS, which apcupsd sets for the killpower signal -- to cause the UPS
     to shut off the power.
```

Please note that these actions apply only to the SIMPLE cable, the signals used on the other cables are different.

Finally, here is another way of looking at the CUSTOM-SIMPLE cable:

```
APCUPSD SIMPLE-CUSTOM CABLE

Computer Side  |  Description of Cable           |     UPS Side
DB9f   |  DB25f |                                 |   DB9m   | DB25m
4      |   20   |  DTR (5vcc)             *below  |    n/c   |
8      |    5   |  CTS (low battery)      *below  | <-  5    |   7
2      |    3   |  RxD (no line voltage)  *below  | <-  3    |   2
5      |    7   |  Ground (Signal)                |     4    |  20
1      |    8   |  CD (on battery from UPS)       | <-  2    |   3
7      |    4   |  RTS (kill UPS power)           | ->  1    |   8
n/c    |    1   |  Frame/Case Gnd (optional)      |     9    |  22

Note: the <- and -> indicate the signal direction.

Optional connections of original SIMPLE-CUSTOM specification
that are not used.

              4.7K ohm
 DTR    4   --[####]--*              Note needed
                      |
 RxD    2   ----------*---------  3  Not used by Apcupsd
```

When using this cable with apcupsd specify the following in apcupsd.conf:

```
UPSCABLE simple
UPSTYPE dumb
DEVICE /dev/ttyS0 (or whatever your serial port is)
```

21.4 Other APC Cables that apcupsd Supports

apcupsd will also support the following off the shelf cables that are supplied by APC

- 940-0020B/C Simple Signal Only, all models.
- 940-0023A Simple Signal Only, all models.
- 940-0119A Simple Signal Only, Back-UPS Office, and BackUPS ES.
- 940-0024[B/C/G] SmartMode Only, SU and BKPro only.
- 940-0095[A/B/C] PnP (Plug and Play), all models.
- 940-1524C SmartMode Only
- 940-0127A/B USB Cables
- 940-0128A Simple Signal Only, Back-UPS CS in serial mode.

21.5 Voltage Signalling Features Supported by Apcupsd for Various Cables

The following table shows the features supported by the current version of apcupsd (3.8.5 or later) for various cables running the UPS in voltage-signalling mode.

Cable	Power Loss	Low Battery	Kill Power	Cable Disconnected
940-0020B	Yes	No	Yes	No
940-0020C	Yes	Yes	Yes	No
940-0023A	Yes	No	No	No
940-0119A	Yes	Yes	Yes	No
940-0127A	Yes	Yes	Yes	No
940-0128A	Yes	Yes	Yes	No
940-0095A/B/C	Yes	Yes	Yes	No
simple	Yes	Yes	Yes	No

21.6 Voltage Signalling

Apparently, all APC voltage-signalling UPSes have the same signals on the output pins of the UPS. The difference at the computer end is due to different cable configurations. Thus, by measuring the connectivity of a cable, one can determine how to program the UPS. This is to be verified.

The signals presented or accepted by the UPS on its DB9 connector using the numbering scheme listed above is:

```
UPS Pin         Signal meaning
 1     <-     Shutdown when set by computer for 1-5 seconds.
 2     ->     On battery power (this signal is normally low but
                    goes high when the UPS switches to batteries).
 3     ->     Mains down (line fail) See Note 1 below.
 5     ->     Low battery. See Note 1 below.
 6     ->     Inverse of mains down signal. See Note 2 below.
 7     <-     Turn on/off power (only on advanced UPSes only)

 Note 1: these two lines are normally open, but close when the
     appropriate signal is triggered. In fact, they are open collector
     outputs which are rated for a maximum of +40VDC and 25 mA. Thus
     the 4.7K ohm resistor used in the Custom Simple cable works
     quite well.

 Note 2: the same as note 1 except that the line is normally closed,
     and opens when the line voltage fails.
```

21.7 The Back-UPS Office 500 signals

The Back-UPS Office UPS has a telephone type jack as output, which looks like the following:

```
Looking at the end of the connector:

   6 5 4 3 2 1
  -------------
 | . . . . . . |
 |             |
 |  |----------|
 |__|
```

It appears that the signals work as follows:

```
  UPS            Signal meaning
1 (brown)    <-   Shutdown when set by computer for 1-5 seconds.
2 (black)    ->   On battery power
3 (blue)     ->   Low battery
4 (red)           Signal ground
5 (yellow)   <-   Begin signalling on other pins
6 (none)          none
```

21.8 Analyses of APC Cables

21.8.1 940-0020B Cable Wiring: This diagram is for informational purposes and is not complete. Although we do not know what the black box semi-conductor contains, we believe that we understand its operation (many thanks to Lazar M. Fleysher for working this out).

This cable can only be used on voltage-signalling UPSes, and provides the On Battery signal as well as kill UPS power. Most recent evidence (Lazar's analysis) indicates that this cable under the right conditions may provide the Low Battery signal. This is to be confirmed.

```
APC Part# - 940-0020B

Signal Computer                  UPS
       DB9F                     DB9M
 CTS    8   --------------------  2  On Battery
 DTR    4   --------------------  1  Kill power
 GND    5   ----------------*----  4  Ground
                            |
                 ---        *----  9  Common
 DCD    1  ----|///|-----------   5  Low Battery
               |\\\|
 RTS    7  ----|///| (probably a
                ---   semi-conductor)
```

Thanks to Lazar M. Fleysher.

21.8.2 940-0020C Cable Wiring: This diagram is for informational purposes and may not be complete, we don't recommend that use it to build you build one yourself. This cable can only be used on voltage-signalling UPSes, and provides the On Battery signal, the Low Battery signal as well as kill UPS power. In apcupsd versions 3.8.2 and prior, please set your UPSCABLE to 940-0020B. In version 3.8.3 and later, you may specify the cable as 940-0020C. Please note that this diagram may not be accurate.

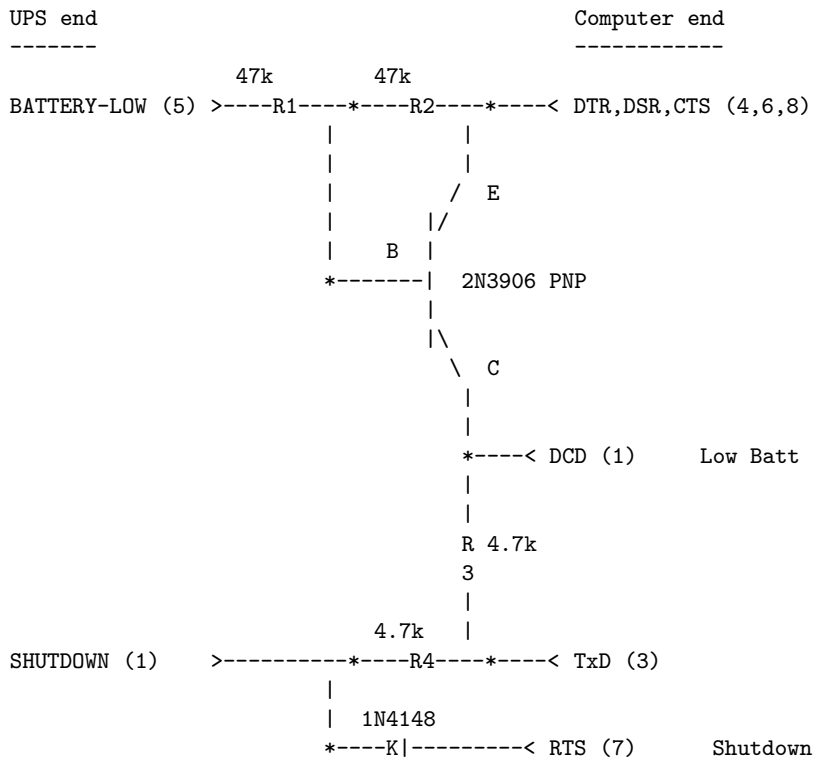
```
APC Part# - 940-0020C

Signal Computer                  UPS
       DB9F                     DB9M
 CTS    8   --------------------  2  On Battery
 DTR    4   --------------------  1  Kill power
 GND    5   ----------------*----  4  Ground
                            |
                            *----  9  Common
 RTS    7 -----[ 93.5K ohm ]-----  5  Low Battery
              or semi-conductor
```

21.8.3 940-0023A Cable Wiring: This diagram is for informational purposes and may not be complete, we don't recommend that use it to build you build one yourself. This cable can only be used on voltage-signalling UPSes, and apparently only provides the On Battery signal. As a consequence, this cable is pretty much useless, and we recommend that you find a better cable because all APC UPSes support more than just On Battery. Please note that we are not sure the following diagram is correct.

```
APC Part# - 940-0023A

Signal Computer                  UPS
       DB9F                     DB9M
 DCD    1   --------------------  2  On Battery

              3.3K ohm
 TxD    3   --[####]-*
                     |
 DTR    4   ---------*
 GND    5   ----------------*----  4  Ground
                            |
                            *----  9  Common
```

21.8.4 940-0095A Cable Wiring: This is the definitive wiring diagram for the 940-0095A cable submitted by Chris Hanson <cph at zurich.ai.mit.edu>, who disassembled the original cable, destroying it in the process. He then built one from his diagram and it works perfectly.

```
Construction and operation of the APC #940-0095A cable.
This cable is included with the APC Back-UPS Pro PNP series.
```



Operation:

* DTR is "cable power" and must be held at SPACE. DSR or CTS may be used as a loopback input to determine if the cable is plugged in.

* DCD is the "battery low" signal to the computer. A SPACE on this line means the battery is low. This is signalled by BATTERY-LOW being pulled down (it is probably open circuit normally).

Normally, the transistor is turned off, and DCD is held at the MARK voltage by TxD. When BATTERY-LOW is pulled down, the voltage divider R2/R1 biases the transistor so that it is turned on, causing DCD to be pulled up to the SPACE voltage.

* TxD must be held at MARK; this is the default state when no data is being transmitted. This sets the default bias for both DCD and SHUTDOWN. If this line is an open circuit, then when BATTERY-LOW is signalled, SHUTDOWN will be automatically signalled; this would be true if the cable were plugged in to the UPS and not the computer, or if the computer were turned off.

```
* RTS is the "shutdown" signal from the computer.  A SPACE on this
line tells the UPS to shut down.

* RxD and RI are both the "power-fail" signals to the computer.  A
MARK on this line means the power has failed.

* SPACE is a positive voltage, typically +12V.  MARK is a negative
voltage, typically -12V.  Linux appears to translate SPACE to a 1
and MARK to a 0.
```

21.8.5 940-0095B Cable Wiring: This diagram is for informational purposes and may not be complete, we don't recommend that use it to build one yourself.

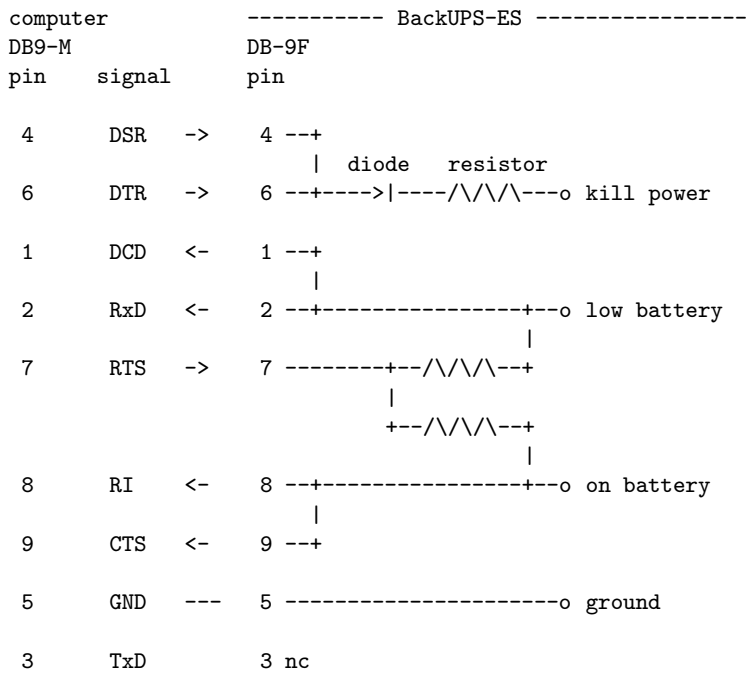
```
APC Part# - 940-0095B

Signal Computer                  UPS
       DB9F                     DB9M
 DTR    4   ----*
 CTS    8   ----|
 DSR    6   ----|
 DCD    1   ----*
 GND    5   ----------------*----  4  Ground
                            |
                            *----  9  Common
 RI     9   ----*
                |
 RxD    2   ----*---------------  2  On Battery
 TxD    3   ----------[####]----  1  Kill UPS Power
                       4.7K ohm
```

21.8.6 940-0119A Cable Wiring: This diagram is for informational purposes and may not be complete, we don't recommend that use it to build you build one yourself. This cable is used with the BackUPS Office UPSes.

```
APC Part# - 940-0119A

  UPS      Computer
  pins     pins      Signal             Signal meaning
1 (brown)    4,6      DSR DTR     <-   Shutdown when set by computer for 1-5 seconds.
2 (black)    8,9      RI  CTS     ->   On battery power
3 (blue)     1,2      CD  RxD     ->   Low battery
```

```
4 (red)       5       Ground
5 (yellow)    7       RTS         <-   Begin signalling on other pins
6 (none)     none
```

21.8.7 BackOffice ES: The BackUPS ES has a straight through serial cable with no identification on the plugs. To make it work with apcupsd, specify the **UPSCABLE 940-0119A** and **UPSTYPE backups**. The equivalent of cable 940-0119A is done on a PCB inside the unit. Thanks to William Stock for supplying us with the information about the straight through cable, the PCB, and the following diagram:

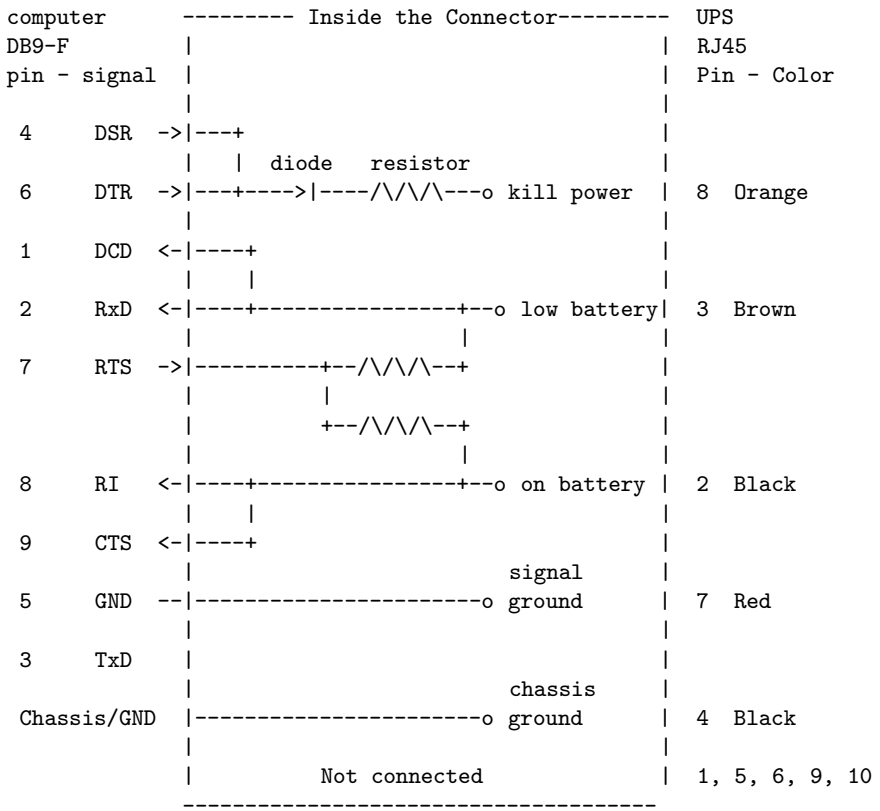


21.8.8 BackUPS ES and CS in Serial mode with Cable 940-0128A: Though these UPSes are USB UPSes, APC supplies a serial cable (typically with a green DB9 F connector) that has 940-0128A stamped into one side of the plastic serial port connector. The other end of the cable is a 10 pin RJ45 connector that plugs into the UPS (thanks to Dean Waldow for sending me a cable!). Apcupsd version 3.8.5 and later supports this cable when specified as **UPSCABLE 940-0128A** and **UPSTYPE backups**. However, running

in this mode much of the information that would be available in USB mode is lost. In addition, when apcupsd attempts to instruct the UPS to kill the power, it begins cycling about 4 times a second between battery and line. The solution to the problem (thanks to Tom Suzda) is to unplug the UPS and while it is still chattering, press the power button (on the front of the unit) until the unit beeps and the chattering stops. After that the UPS should behave normally and power down 1-2 minutes after requested to do so.

An amazing discovery by slither_man allows one to build a CUSTOM-RJ45 cable (documented above) and run the BackUPS CS (and probably also the ES) in Smart mode. Running it this way provides all the same information that you would get by running it in USB mode. As a consequence, we recommend that you either purchase (where I don't know) or build your own CUSTOM-RJ45 cable rather than use the 940-0128A cable.

Thanks to all the people who have helped test this and have provided information on the cable wiring, our best guess for the cable schematic is the following:

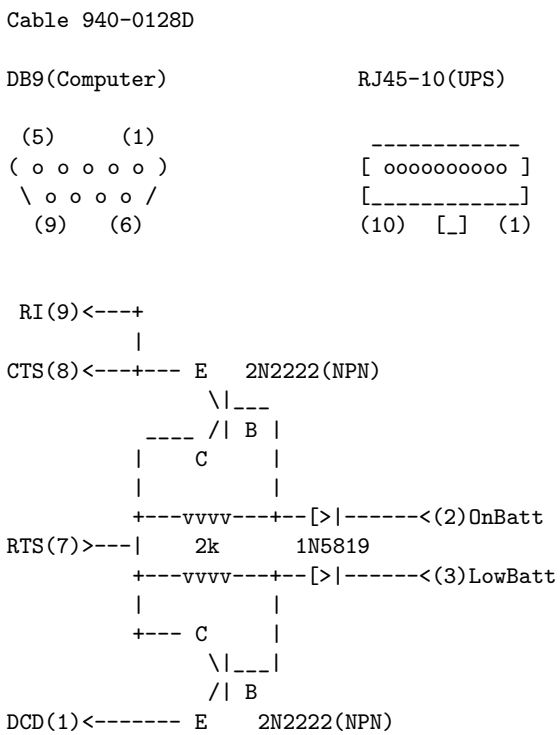


```
The RJ45 pins are: looking at the end of the connector:

10 9 8 7 6 5 4 3 2 1
-----------------------
| . . . . . . . . . . |
|                     |
-----------------------
       |____|
```

From Jan Babinski jbabinsk at pulsarbeacon dot com: (many thanks)

The BackUPS XS1000(BX-1000) ships with the 940-0127B usb cable and the optional 940-0128D serial cable. I'm trying to get it running under OpenBSD so usb is out of the question. I've tried smart serial with the custom cable, but that's not working, so I'm stuck with dumb serial over 940-0128D.

I have noticed that this cable is not supported by apcupsd so I set out to find out how much different this cable is compared with the supported ones. After tearing the cable apart (see schematic below) I noticed that it is functionally similar to the 940-0128A cable except for NC on (6) DTR and (2) RD on the computer side.



```
DTR(4)>--------------------------->(8)KillPwr

GND(5)-----------------------------(7)Signal GND
(Shield)---------------------------(4)Chassis GND


Cable 940-0127B

USB(Computer)      RJ45-10(UPS)
 ---------          ------------
| = = = = |        [ ooooooooo ]
|_________|        [____________]
 (1)   (4)         (10)  [_]  (1)

  +5V(1)-----------(1)+5V
DATA+(2)-----------(9)DATA+
DATA-(3)-----------(10)DATA-
  GND(4)-----------(7)Signal GND
(Shield)-----------(4)Chassis GRND
```

21.9 Win32 Implementation Restrictions for Simple UPSes

Due to inadequacies in the Win32 API, it is not possible to set/clear/get all the serial port line signals. apcupsd can detect: CTS, DSR, RNG, and CD. It can set and clear: RTS and DTR.

This imposes a few minor restrictions on the functionality of some of the cables. In particular, LineDown on the Custom Simple cable, and Low Battery on the 0023A cable are not implemented.

21.10 Internal Apcupsd Actions for Simple Cables

```
This section describes how apcupsd 3.8.5 (March 2002)
treats the serial port line signals for simple cables.

apcaction.c:
 condition = power failure detected
 cable = CUSTOM_SIMPLE
 action = ioctl(TIOCMBIS, DTR)      set DTR (enable power bit?)

apcaction.c:
 condition = power back
 cable = CUSTOM_SIMPLE
 action = ioctl(TIOCMBIC, DTR)      clear DTR (clear power bit)
```

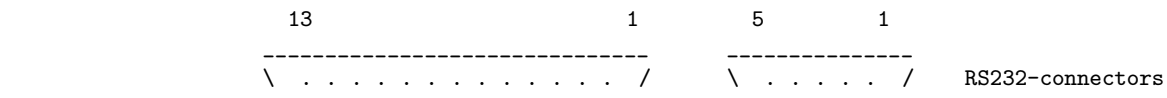
```
 action = ioctl(TIOCMBIC, ST)       clear ST (TxD)

apcserial.c:
 condition = serial port initialization
 cable = 0095A, 0095B, 0095C
 action = ioctl(TIOMBIC, RTS)       clear RTS (set PnP mode)

 cable = 0119A, 0127A, 0128A
 action = ioctl(TIOMBIC, DTR)       clear DTR (killpower)
 action = ioctl(TIOMBIS, RTS)       set   RTS (ready to receive)

apcserial.c:
 condition = save_dumb_status
 cable = CUSTOM_SIMPLE
 action = ioctl(TIOMBIC, DTR)       clear DTR (power bit?)
 action = ioctl(TIOMBIC, RTS)       clear RTS (killpower)

 cable = 0020B, 0020C, 0119A, 0127A, 0128A
 action = ioctl(TIOMBIC, DTR)       clear DTR (killpower)

 cable = 0095A, 0095B, 0095C
 action = ioctl(TIOMBIC, RTS)       clear RTS (killpower)
 action = ioctl(TIOMBIC, CD)        clear DCD (low batt)
 action = ioctl(TIOMBIC, RTS)       clear RTS (killpower) a second time!

apcserial.c:
 condition = check_serial

 cable = CUSTOM_SIMPLE
 action = OnBatt = CD
 action = BattLow = CTS
 action = LineDown = SR

 cable = 0020B, 0020C, 0119A, 0127A, 0128A
 action = OnBatt = CTS
 action = BattLow = CD
 action = LineDown = 0

 cable = 0023A
 action = Onbatt = CD
 action = BattLow = SR
 action = LineDown = 0

 cable = 0095A, 0095B, 0095C
 action = OnBatt = RNG
 action = BattLow = CD
 action = LineDown = 0


apcserial.c
 condition = killpower

 cable = CUSTOM_SIMPLE, 0095A, 0095B, 0095C
 action = ioctl(TIOMCBIS, RTS)      set RTS (kills power)
```

```
action = ioctl(TIOMCBIS, ST)       set TxD

cable = 0020B, 020C, 0119A, 0127A, 0128A
action = ioctl(TIOMCBIS, DTR)      set DTR (kills power)
```

21.11 RS232 Wiring and Signal Conventions

DB-25 Pin #	DB-9 Pin #	Name	DTE-DCE Description
1	—	FG	– Frame Ground/Chassis GND
2	3	TD	–> Transmitted Data, TxD
3	2	RD	<– Received Data, RxD
4	7	RTS	–> Request To Send
5	8	CTS	<– Clear To Send
6	6	DSR	<– Data Set Ready
7	5	SG	—– Signal Ground, GND
8	1	DCD	<– Data Carrier Detect
9	—	—	– Positive DC test voltage
10	—	—	– Negative DC test voltage
11	—	QM	<– Equalizer mode
12	—	SDCD	<– Secondary Data Carrier Detect
13	—	SCTS	<– Secondary Clear To Send
14	—	STD	–> Secondary Transmitted Data
15	—	TC	<– Transmitter (signal) Clock
16	—	SRD	<– Secondary Receiver Clock
17	—	RC	–> Receiver (signal) Clock
18	—	DCR	<– Divided Clock Receiver
19	—	SRTS	–> Secondary Request To Send
20	4	DTR	–> Data Terminal Ready
21	—	SQ	<– Signal Quality Detect
22	9	RI	<– Ring Indicator
23	—	—	–> Data rate selector
24	—	—	<– Data rate selector
25	—	TC	<– Transmitted Clock

21.12 Pin Assignment for the Serial Port (RS-232C), 25-pin and 9-pin, Female End



```
 \  . . . . . . . . . . . . .  /       \  . . . .  /     looking into the
  ---------------------------         -----------      end of the cable.
  25                        14          9       6
```

The diagram above represents the Female end of the cable. The male end is the same, but looking from inside the cable.

```
DTE : Data Terminal Equipment (i.e. computer)
DCE : Data Communications Equipment (i.e. UPS)
RxD : Data received; 1 is transmitted "low", 0 as "high"
TxD : Data sent; 1 is transmitted "low", 0 as "high"
DTR : DTE announces that it is powered up and ready to communicate
DSR : DCE announces that it is ready to communicate; low=modem hang-up
RTS : DTE asks DCE for permission to send data
CTS : DCE agrees on RTS
RI  : DCE signals the DTE that an establishment of a connection is attempted
DCD : DCE announces that a connection is established
```

21.13 Ioctl to RS232 Correspondence

```
#define TIOCM_LE        0x001
#define TIOCM_DTR       0x002
#define TIOCM_RTS       0x004
#define TIOCM_ST        0x008
#define TIOCM_SR        0x010
#define TIOCM_CTS       0x020
#define TIOCM_CAR       0x040
#define TIOCM_RNG       0x080
#define TIOCM_DSR       0x100
#define TIOCM_CD        TIOCM_CAR
#define TIOCM_RI        TIOCM_RNG
#define TIOCM_OUT1      0x2000
#define TIOCM_OUT2      0x4000
```

22 Testing Serial-Line UPSes

If you have a serial-line UPS, there are some tests you should run before the general ones described in the Testing (see Testing Apcupsd) section.

To test your computer's connection with a serial-line UPS, you first need to establish that the serial line is functioning, and then that the UPS is responding to commands. This can be a bit tricky, especially with a dumb voltage-signalling interface, because it is completely quiescent when there are no commands being passed, and the command repertoire doesn't include any self-tests.

Because it is easy to configure a serial cable incorrectly in such a way as to cause premature shutdowns of the UPS power, we *strongly* recommend, especially for voltage- signaling (dumb) UPSes, that you do most of the initial testing with your computer plugged into the wall rather than your UPS. Thus if the UPS power is suddenly shut off, your computer will continue to run. We also recommend using safe-apccontrol as described below, until you are sure that the signaling is correct.

Also note that if you launch the execution of apcupsd while your voltage-signaling UPS is on battery power, it is very likely that your UPS will immediately shut off the power. This is due to the initialization of the serial port line signals, which often looks to the UPS like a shutdown command.

Finally, double-check the state of your cabling and UPS indicator lights frequently during testing. For voltage-signaling UPSes, apcupsd is not currently able to detect whether or not the serial cable is connected. In addition, some simple signaling UPSes with certain cable combinations are not able to detect the low battery condition. For more details please see Voltage Signalling Features Supported by Apcupsd for Various Cables.

22.1 Establishing Serial Port Connection

Once you have compiled, installed, and invoked apcupsd, you should wait to allow apcupsd to configure itself and establish contact with the UPS.

If you see the following message about 30 seconds after starting apcupsd:

```
apcupsd FATAL ERROR in apcserial.c at line 156
PANIC! Cannot communicate with UPS via serial port.
```

it means that apcupsd tried for about 30 seconds to establish contact with the UPS via the serial port, but was unable to do so. Before continuing, you must correct this problem. Some of the possible sources of the problem are:

- You have not configured the correct serial port name on the DEVICE directive in your apcupsd configuration file.
- The serial port that you have chosen has logins enabled. You must disable logins on that port, otherwise, the system prevents apcupsd from using it. Normally, the file /etc/inittab specifies the ports for which a getty process is started (on Sun machines, the serial port program equivalent to getty is called ttymon). You must disable this for the port that you wish to use.

- Make sure you are doing your testing as **root** otherwise, you may have permissions problems accessing the serial port.
- You may have cabling problems, either with an incorrect cable, or the incorrect cable specification directive in the configuration file.
- You may have a problem with the /etc/apcupsd/acpupsd.conf file. For example, check that you have specified the correct type of UPS and the correct networking directives. For more details, see the After Installation.
- If you have a SmartUPS 5000 RM 15U or similar model, that comes with a "Web/SNMP management card" in one of the "Smart Slots", this card may interfere with the serial port operation. If you are having problems, please remove this card and try again. Supposedly V3.0 of the card firmware has been corrected to properly release the serial port.
- Ensure that you have no other programs that are using the serial port. One user reported that he had problems because the serial port mouse (gpm) was using the same port as apcupsd. This causes intermittent seemingly random problems.
- If you are using a WinNT or Win2000 machine, the OS is probably attempting to attach a serial mouse to the port you are using (COM1 or COM2). To prevent this, edit your c:\boot.ini file, and you will find a line that looks something like the following:

```
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 4.00"
```

  Add the following to the end of the line: /NoSerialMice:COM1 (or COM2) so that the new line looks like:

```
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows NT Workstation Version 4.00" /NoSerialM
```

- If you are using a WinNT or Win2000 machine, try connecting apcupsd to COM2 rather than COM1 (be sure to change your c:\apcupsd\etc\apcupsd\apcupsd.conf to reflect the change).
- If you are using a Solaris machine, you may have similar problems as described above for the WinNT machine. A possible fix is documented in the Sun section of the Configuration chapter of this manual.

- Try connecting your UPS to another machine. If it works, then you probably have a bad serial port card. As unlikely as this may sound, at least two of our users have had to replace bad serial port cards.
- Try doing an `lsof /dev/ttyS0` where you replace the **/dev/ttyS0** with your serial port name. If you get no output, the port is free (or there is no physical port). If you get output, then another program is using the port, and you should see which one.
- Try doing a `dmesg | grep tty`. This may show you if a program has grabbed the port. (Thanks to Joe Acosta for the suggestion.)
- If all else fails, make sure your system is configured for serial port support.
- If you are running Linux, check your /proc file system. For example: `cat /proc/devices` should print something like *4 ttyS* if you have a serial port. If your serial port is working, a `cat /proc/interrupts` should show the serial port usage (e.g. **4: 294553 XT-PIC serial**) Also, `cat /proc/ioports` should show up something like **03f8-03ff : serial(auto)**. Or, `cat /proc/tty` should print a line like **serial /dev/ttyS 4 64-127 serial**. Finally, a `cat /proc/tty/driver/serial` should print something like the following:

```
serinfo:1.0 driver:5.05c revision:2001-07-08
0: uart:16550A port:3F8 irq:4 baud:9600 tx:1503168 rx:1461721 fe:8
```

The first thing to do is to look at your log file, usually /var/log/messages because apcupsd writes more detailed information to the log file whenever there is an error.

If you have a UPS that uses apcsmart protcol (see table of types (see type_table) for a list of the UPSes using these protocols), you can manually test the serial communications with the UPS by starting a serial port communications program (such as minicom, tip, or cu) with the settings 2400 8N1 (2400 baud, 8 bits, no parity, 1 stop bit). Be extremely careful what you send to your UPS as certain characters may cause it to power down or may even cause damage to the UPS. Try sending an upper case Y to the UPS (without a return at the end). It should respond with SM. If this is not the case, review the possible problems listed above. If you fat finger the Y and enter y instead, no cause for alarm, you will simply get the APC copyright notice.

Once you are sure that serial port communications is working, proceed to the next test.

22.2 Using apctest on Serial-Line UPSses

On an apcsmart serial-line UPS, apctest will give you access to the battery of low-level tests we described in apctest. If you have a voltage-signalling UPS, it enables a different test repertoire which is described here, Among other things, if you are uncertain about what kind of cable you have, you may be able to use apctest to figure that out.

Shutdown apcupsd if it is running. Make sure your /etc/apcupsd/apcupsd.conf file has **UPSTYPE backups** and **UPSCABLE simple** Normally apctest will have been built and installed by default, otherwise, you can explicitly build it on Unix with:

```
cd <apcupsd-source-directory>
make apctest
./apctest
```

on Win32 systems, use:

```
make apctestwin32
./apctest
```

It will present you with the following output

```
2001-02-07 04:08:26 apctest 3.8.5 (3 January 2002) redhat
Checking configuration ...
sharenet.type = DISABLE
cable.type = CUSTOM_SIMPLE
mode.type = BK
Setting up serial port ...
Creating serial port lock file ...
Doing prep_serial() ...
Hello, this is the apcupsd Cable Test program.
This part of apctest is for testing dumb UPSes (ones that uses signaling rather than commands.
Most tests enter a loop polling every second for 10 seconds.
```

Then it will present you with the following list of choices:

```
1) Test 1 - normal mode
2) Test 2 - no cable
```

```
3) Test 3 - no power
4) Test 4 - low battery (requires test 3 first)
5) Test 5 - battery exhausted
6) Test 6 - kill UPS power
7) Test 7 - run tests 1 through 5
8) Guess which is the appropriate cable
9) quit

Select test number:
```

Run tests 1, 2, and 3. Note, none of the currently supported cables will indicate a change for test 2. You can then run test 8 to see what cable it thinks you should be using. Finally run test 4.

apctest can also be run for Smart UPSes.

The print out of your testing will be written to the file apctest.output. If you are unable to solve your problem, you can try posting that file to the development mailing list, and perhaps we can help you. In this case, please also include information on your operating system, which version of apcupsd you are using, your UPS model, and also your apcupsd.conf file.

22.2.1 Expected apctest Signals for a UPS: If you have configured your UPS as:

```
UPSTYPE backups
UPSCABLE APC_940_0119A
    or APC_940_0127A
    or APC_940_0128A
    or APC_940_0020B
    or APC_940_0020C
```

here are typical signals you would expect to see in the output from the various tests of apctest:

```
Test 1 normal:              RTS for cables (0119A 0127A 0128A)
Test 2 no serial cable:     not important
Test 3 no AC power:         CTS for all cables
Test 4 batteries exhausted: CTS and CD for all cables
```

Note: **RTS** if set in Test 1 will probably also be set in all the other tests. This is not important, what counts is the appearance of **CTS** when the power fails and additionally **CD** when the batteries are low.

22.2.2 Expected apctest Signals for a BackUPS Pro: If you have configured your UPS as:

```
UPSTYPE backupspro
UPSCABLE APC_940_0095A
    or APC_940_0095C
```

here are the typical signals you would expect to see in the output from the various tests of apctest:

```
Test 1 normal:              RTS not set
Test 2 no serial cable:     not important
Test 3 no AC power:         RNG
Test 4 batteries exhausted: RNG and CD
```

Note: **RTS** should never be set in any of the tests as it is the killpower signal. What is important is the appearance of **RNG** when the power fails and additionally **CD** when the batteries are low.

23 Troubleshooting Serial Line communications

23.1 Determining Which Voltage-Signaling Cable You Have

The most frequently encountered problem with voltage-signalling UPSes (e.g. BackUPS 650) is that you have incorrectly specified which cable is being used. All cables furnished by APC have the cable number stamped on the side of the computer connector end of the cable. Using this number with apcupsd will normally work fine. If you do not know what cable you have, you can use the apctest program to determine the type of the cable.

For simple signaling UPSes, you should *not* use **simple** in the cable specification (i.e. **UPSCABLE simple**) unless you have made the cable yourself according to the wiring diagram given in the cables chapter of this manual.

23.2 Once you have established serial communications

Once you have established that apcupsd can talk to the UPS over the serial part, go do the series of functional tests described in the main Testing (see Testing Apcupsd) section.

One additional note applies:

23.2.1 Bizarre Intermittent Behavior: In one case, a user reported that he received random incorrect values from the UPS in the status output. It turned out that gpm, the mouse control program for command windows, was using the serial port without using the standard Unix locking mechanism. As a consequence, both apcupsd and gpm were reading the serial port. Please ensure that if you are running gpm that it is not configured with a serial port mouse on the same serial port.

24 Recalibrating the UPS Runtime

Note: In a future release of apcupsd this procedure will be replaced by a daemon operation that can be performed on all types of UPS.

This section does not apply to voltage-signalling or dumb UPSes such as the older BackUPS models.

Smart UPSes internally compute the remaining runtime, and apcupsd uses the value supplied by the UPS. As the batteries age (after say two or three years), the runtime computation may no longer be accurate since the batteries no longer hold the same charge. As a consequence, in the event of a power failure, the UPS and thus apcupsd can report a runtime of 5 minutes remaining when in fact only one minute remains. This can lead to a shutdown before you might expect it, because regardless of the runtime remaining that is reported, the UPS will always correctly detect low batteries and report it, thus causing apcupsd to correctly shutdown your computer.

If you wish to have the UPS recalibrate the remaining runtime calculations, you can do so manually as the current version of apcupsd does not support this feature. To do so,

- Shutdown apcupsd
- contact your UPS directly using some terminal program such as minicom, tip, or cu with the settings 2400 8N1 (2400 baud, 8 bits, no parity, 1 stop bit). Be extremely careful what you send to your UPS as certain characters may cause it to power down or may even cause damage to the UPS. Try sending an upper case Y to the UPS (without a return at the end). It should respond with SM. If this is not the case, read the chapter on testing. If you fat finger the Y and enter y instead, no cause for alarm, you will simply get the APC copyright notice.
- when you are sure you are properly connected send an upper case D (no cr). This will put the UPS into calibration mode, and it will drain

the battery down to 25% capacity (35% for a Matrix) at which point it will go back on the mains. In doing so, it will recompute the runtime calibration.

- If you wish to abort the calibration, enter a second D command.
- When you are done, restart apcupsd.

In principle, you should be able to do this with the computer powered by the UPS, but if you wish to be completely safe, you should plug your computer into the wall prior to performing the runtime calibration. In that case, you will need to artificially load the UPS with light bulbs or other means. You should supply a load of about 30 to 35% but not more than 50%. You can determine the load by looking at the output of the `apcaccess status` command while apcupsd is running.

You should not run the recalibration command more than once or twice per year as discharging these kinds of batteries tends to shorten their life span.

24.1 Status Logging On Serial-Line UPSes

Serial-line UPSes that speak the apcsmart protocol log all of the events described in the Status Format (see apcupsd Status Logging) section of the Technical Reference. Voltage-signalling UPSes, on the other hand, have a much narrower data channel. They can only report a small handful of conditions.

The following summarizes (rather sketchily, sorry) the data you can expect to get from this obsolete hardware. All corrections and additions will be welcome.

From BackUPS Pro and SmartUPS v/s:

```
LINEFAIL : OnlineStatus
BATTSTAT : BatteryStatus
MAINS    : LineVoltageState
LASTEVNT : LastEventObserved
```

BackUPS and NetUPS Simple Signals

```
LINEFAIL : OnlineStatus
BATTSTAT : BatteryStatus
```

25 DATA Logging

This feature is somewhat outdated and not often used.

Data logging consists of periodically logging important data concerning the operation of the UPS. For the definitive definition of the format, see log_data() in apcreports.c. The format varies according to the UPS model and the information available from the UPS.

For UPS models, NBKPRO, SMART, SHARESMART, and MATRIX, the output is written in a format very similar to what PowerChute writes. That is:

MinLineVoltage, MaxLineVoltage, OutputVoltage, BatteryVoltage, LineFrequency, LoadPercent, UPSTemperature,AmbientTemperature,Humidity,LineVoltage, BatteryCharge,toggle

Any value that is not supported by your UPS such as AmbientTemperature and Humidity will be blank or possibly as 0.0. In any case the commas before and after that field will still be output. The toggle value alternates from 0 to 1 on each line. This was added at user request so that no two adjacent samples are identical.

An actual example from the log file is:

```
Nov  2 12:43:05 matou apcupsd[23439]: 224.9,227.5,226.2,27.74,50.00,100.0,30.6,,,226.2,50.0,1
```

Technical Reference

26 Configuration Directive Reference

Configuration directives in /etc/apcupsd/apcupsd.conf control the behavior of the apcupsd daemon. For most installations it is only necessary to set a handful of general directives. The rest can be left at their defaults unless you have an exotic configuration.

26.1 General Configuration Directives

In general, each of these directives is required (ecept that the DEVICE directive is ignored for UPSCABLE ether).

UPSTYPE <type of APC UPS you have> The name of a driver. Should be one of dumb, apcsmart, net, usb, snmp, or test. This describes your interface type.

The UPSTYPE directive can be defined during installation by using the `--with-upstype=` option of the `./configure` program.

UPSCABLE <type of cable you are using>>

```
[ simple | 940-0020B | 940-0023A ]
[ smart | 940-0024B | 940-0024C ]
[ 940-1524C | 940-0024G | 940-0095A | 940-0095B | 940-0095C | 940-0119A]
[ ether | usb ]
```

The `--with-upscable=` option of `./configure` can be used to set a default for this directive during the your build.

DEVICE <name of device> Specify which device is used for UPS communications. For serial ports, it is usually something like /dev/ttyS0. For USB ports, you may leave the name of the device blank (no specification) and apcupsd will automatically search the standard locations for the UPS.

Normally, the `./configure` program will set an appropriate default value. You may also specify the `--with-serial-dev=` option of the `./configure` program to set this directive at build time.

If you have specified **UPSTYPE net**, then the device name to be specified consists of **hostname:port** where the hostname is the fully qualified name or IP address of the host (NIS server) and the port (optional) is the port to use to contact the server.

If you specified **UPSTYPE snmp**, then the device name becomes **hostname:vendor:community**. Please see the SNMP chapter (see Support for SNMP UPSes) in this manual for more details.

LOCKFILE <path to lockfile> This option tells apcupsd where to create a lockfile for the USB or serial port in the specified directory. This is important to keep two programs from reading or writing the port at the same time. Please note that although the directive name is LOCKFILE, you are actually specifying the lock file path. apcupsd automatically appends the name of the device when creating the file. On most systems, this directive is automatically set by the `./configure` program. You may also explicitly set it during the build process by using the `--with-lock-dir=` option of the `./configure` program.

26.2 Configuration Directives Used by the Network Information Server

None of these directives are required for proper operation of apcupsd. For the Network Information Server to work, it must be enabled in the configuration (default) with `--enable-nis`

NETSERVER [**on** — **off**] This configuration directive turns the network information server on or off. If it is on, apcupsd will spawn a child process that serves STATUS and EVENTS information over the network. This information is currently used by the Web-based CGI programs. The default is on. In some cases, for added security, you may want to invoke a separate information server daemon from the inetd daemon. In that case, *NETSERVER* should be *off*.

NISIP <**IP-address**> This directive specifies an IP address on which NIS server will listen for incoming connections. Default value is 0.0.0.0 that means any incoming request will be serviced but if you want it to listen to a single subnet you can set it up to that subnet address, for example 192.168.10.0. Additionally you can listen for a single IP like 192.168.10.1. You may also use the `--with-nisip=` option of the `./configure` program to set this directive during the build.

This directive does not work on Win32 machines because inet_ipton() is not implemented there.

NISPORT <**port**> This configuration directive specifies the port to be used by the apcupsd Network Information Server. The default is platform dependent, but typically 3551, which we have received from IANA as the official apcupsd networking port. If you change this port, you must manually change the #define SERV_TCP_PORT in cgi/upsfetch.c and rebuild the CGI programs. An alternative is to use the `--with-nis-port=` option of the `./configure` program during the build. In this case, all the appropriate locations will be automatically changed.

EVENTSFILE <**filename**> If you want the apcupsd network information server to provide the last 10 events via the network, you must specify a file where apcupsd will save these events. The default is: /etc/apcupsd/apcupsd.events. Currently, apcupsd will save at most the last 50 events. Periodically (once an hour by default), apcupsd will check the size of this file. When more than 50 events are recorded, apcupsd will truncate the file to the most recent 10 events. Consequently this file will not grow indefinitely. Although we do not recommend it, you may change these values by editing apcevents.c and

changing the appropriate defines. Be aware that if you set these values to very large numbers, apcupsd may make excessive memory demands on the system during the data access and file truncation operations.

This filename may also be specified at build time by using the `--with-log-dir=` option of the `./configure` program.

26.3 Configuration Directives used during Power Failures

In general, none of these directives are required. However, if you have a voltage-signalling (dumb) UPS with a cable that does not support the Low Battery signal, you must set the TIMEOUT directive to force a shutdown. Please see the Cables (see Cables) section of this manual for more details.

ANNOY <time in seconds> Specify the time in seconds between messages requesting logged in users to get off the system during a power failure. This timer starts only when the UPS is running on batteries. The default is 300 seconds (5 minutes). apcupsd sends the annoy messages by invoking the apccontrol script with the `annoyme` argument. The default is to send a wall message on Unix systems and a popup message in Windows.

The value of **ANNOYDELAY** must be greater than the value of **ANNOY** in order to receive annoy messages (this doesn't make sense, and means that the default values do not generate annoy messages: KES).

Note that if **NOLOGON** is set to **disable** the annoy messages will also be disabled.

ANNOYDELAY <time in seconds> Specify delay time in seconds before apcupsd begins requesting logged in users to get off the system during a power failure. This timer starts only after the UPS is running on batteries. This timer is reset when the power returns. The default is 60 seconds. Thus, the first warning to log off the system occurs after 60 seconds on batteries, assuming that **NOLOGON** is not set to **disable**.

NOLOGON <specifies when apcupsd should prevent user logins> [disable — timeout — percent — minutes — always] are valid types.

The type specified allows you define the point when apcupsd will create the /etc/nologin file and thus when user logins are prohibited. Once the /etc/nologin file is created, normal users are prevented from logging in. Control of when this file is created is important for allowing

systems with big UPSes to run as normally until the system administrator determines the need for preventing user logins. The feature also allows the system administrator to hold the "ANNOY" factor until the /etc/nologin file is created. The default is always if no NOLOGON directive is specified.

As far as I can tell, the only useful types are disable and always since the difference in the time when the logout warning is given and shutdown occurs for the other types is very short (KES).

disable prevents apcupsd from creating the nologin file. Consequently, any user can login during a power failure condition. Also, the ANNOY feature is disabled so users will not be warned to logoff the system.

timeout specifies that apcupsd should prohibit logins after the UPS is on batteries for 90% of the time specified on the **TIMEOUT** configuration directive. Note! Normally you don't want to specify a TIMEOUT value, so this option is probably not too useful (KES).

percent specifies that apcupsd should prohibit logins when the remaining battery charge percentage reaches 110% or less than the value specified on the **BATTERYLEVEL** configuration directive. Thus if the **BATTERYLEVEL** is specified as 15, apcupsd will prohibit logins when the battery charge drops below 16% (15% X 110% = 16%).

minutes specifies that apcupsd should prohibit logins when the remaining runtime in minutes reaches 110% or less than the value specified on the **MINUTES** configuration directive. Thus if **MINUTES** is set to 3, apcupsd will prohibit logins when the remaining runtime is less than 3 minutes (3 X 110% = 3).

always causes apcupsd to immediately prohibit logins when a power failure occurs. This will also enable the ANNOY feature.

BATTERYLEVEL <percent of battery> If BATTERYLEVEL is specified, during a power failure, apcupsd will halt the system when the remaining battery charge falls below the specified percentage. The default is 5 percent. This directive is ignored for dumb (voltage-signalling) UPSes. To totally disable this counter, set **BATTERYLEVEL -1** in your apcupsd.conf file.

MINUTES <battery runtime in minutes> If MINUTES is specified, during a power failure, apcupsd will shutdown the system when the remaining runtime on batteries as internally calculated by the UPS falls below the time specified. The default is 3. This directive is

ignored for dumb (voltage-signalling) UPSes. It should be noted that some UPSes report an incorrect value for remaining runtime when the battery is fully charged. This can be checked by examining the **TIMELEFT** value as printed in the output of an `apcaccess status` command. If the value is zero or otherwise unreasonable, your UPS is probably broken. In this case, we recommend that you disable this timer by setting **MINUTES -1** in your apcupsd.conf file.

TIMEOUT <time in seconds> After a power failure, apcupsd will halt the system when TIMEOUT seconds have expired. A value of zero disables this timer. Normally for all Smart UPS models and dumb UPSes with cables that support low battery detection, this should be zero so that the shutdown time will be determined by the battery level and/or remaining runtime (see above) or in the case of a voltage-signalling UPS, when the battery is exhausted. This command is required for dumb UPSes that do not provide a battery exhausted signal (only testing can determine this point). For more information, see the Testing (see

Testing Apcupsd) section of this manual. This timer can also be useful if you want some slave machines to shutdown before other machines to conserve battery power. It is also useful for testing apcupsd because you can force a rapid shutdown by setting a small value (e.g. 60) and pulling the plug to the UPS.

When apcupsd is running in master mode (UPSCLASS netmaster), and a shutdown condition is determined, apcupsd will notify each of the slaves to perform a shutdown then apcupsd will sleep for 30 seconds before issuing the shutdown of its own computer. If you need the master to wait additional time before shutting down (to allow for shutdown of slower slaves or of slaves running software that requires more time to shutdown – e.g. databases), you can do so by adding additional sleep() commands to /etc/apcupsd/apccontrol in each case that causes a shutdown.

TIMEOUT, **BATTERYLEVEL**, and **MINUTES** can be set together without problems. apcupsd will react to the first case or test that is valid. Normally SmartUPS users will set **TIMEOUT** to zero so that the system is shutdown depending on the percentage battery charge remaining (**BATTERYLEVEL**) or the remaining battery runtime (**MINUTES**).

KILLDELAY <time in seconds> If killdelay is set, apcupsd will continue running after a shutdown has been requested, and after the specified time in seconds, apcupsd will attempt to shut off the UPS the power. This directive should normally be disabled by setting the value to zero, but on some systems such as Win32 systems apcupsd cannot

regain control after a shutdown to force the UPS to shut off the power. In this case, with proper consideration for the timing, the **KILLDELAY** directive can be useful. Please be aware, if you cause apcupsd to kill the power to your computer too early, the system and the disks may not have been properly prepared. In addition, apcupsd must continue running after the shutdown is requested, and on Unix systems, this is not normally the case as the system will terminate all processes during the shutdown.

26.4 Configuration Directives used to Control System Logging

STATTIME<time> This directive supplies the time interval between writes to the STATUS file. If set to zero, the STATUS file will not be written. Please note that in a future version of apcupsd the STATUS file code will disappear since its functionality has been replaced by the Network Information Server and by **apcaccess status**, as a consequence, it is normally disabled by setting it to zero.

STATFILE <file> This directive specifies the file to be used when writing the STATUS information. The default is /etc/apcupsd/apcupsd.status.

DATATIME<time> This directives supplies the time interval between writes of PowerChute&tm; like data information to the log file. See the Data (see DATA Logging) section of this manual for additional details.

FACILITY<log-facility> The facility directive can be used to change the system logging class or facility. The default is **DAEMON**. This parameter can be useful if you wish to direct the apcupsd system logging information to other than your system default files. See the Logging section of this manual for additional details.

26.5 Configuration Directives for Sharing a UPS

The following directives apply to the master/slave networking mode of apcupsd where multiple machines can be powered by the same UPS. One machine, the master, will have a serial port connection to the UPS, and the other machines, the slaves, will obtain their information via the network from the master.

Note, as of version 3.10.x, the old master/slave code is by default turned off in the configuration. You must explicitly enable it by including a `--enable-master-slave` option on your `./configure` command before building the source.

In addition to the old master/slave code, there is now a new network driver enabled with `--enable-net` (default disabled) that can be used to control a slave from any version of apcupsd running NIS. This is a much more flexible system of controlling slaves because a slave machine that also has NIS turned on can thus act as a master for another slave with `--enable-net` turned on. With this mode turned on, the slave obtains the address of the master from the **DEVICE** directive, which takes the form **hostname[:port]** as a consequence, none of the directives apply for this form of networking. In addition, for this mode to work, you must specify **UPSTYPE net** so that the proper driver is loaded.

The remainder of this section presents directives that apply to the old master/slave code that must be enabled by the **enable-master-slave** configuration option.

UPSCLASS <class of operation> [standalone — shareslave — sharemaster] and [netslave — netmaster] are valid types. [standalone — netslave — netmaster] are tested classes. [shareslave — sharemaster] classes are being tested.

The default is **standalone** and should be used for all machines powered by the UPS and having a serial port connection to the UPS, but where there are no other computers dependent on power from the same UPS. This is the normal case.

Use **netmaster**, if and only if you have a serial port connection to the UPS and there are other machines deriving power from the same UPS. This is required in all master configuration files.

Use **netslave** if and only if you have no serial port connection to the UPS, but you derive power from it. This is required in all slave configuration files, and in this case, you will also have UPSCABLE set to ether.

Use **shareslave** if and only if you are using a ShareUPS and connected to a BASIC Port with Simple Signal. This code is not fully tested.

Use **sharemaster**, if and only if you are using a ShareUPS and connected to the ADVANCED Port Smart Signal control. This code is not fully tested.

UPSMODE [disable — share — net — sharenet] are valid types. [disable — net] are the only known and tested classes.

[share — sharenet] classes are being tested.

For normal standalone operations, you will set UPSMODE to disable to indicate that you are disabling the master/slave networking.

However, if you are using a single UPS to power several computers and you have configured master and slave computers, then set this value to net.

Use share for two or seven (2/7) additional simple signal ports on a SmartAccessories(tm) (internal/external box) for SmartUPSes. The share and sharenet code is not fully tested.

NETTIME <time in seconds> The interval in seconds that the master uses to send information to slave machines. This rate is automatically set to 1 second if the UPS goes on batteries and reset to your specified value when the mains power returns. A typical value might be 60 seconds.

NETPORT <IP port number> This port number is used for communications in the master/slave networking code. Note that the master and each slave must have the same port number specified on the NETPORT directive in the configuration file. This port may also be specified during installation by using the `--with-net-port=` option of the `./configure` program.

The NETPORT should not be confused with the port number for the Network Information Server which is specified with the **SERVERPORT** configuration directive.

MASTER <name of the master> for slave machines. Used in slave configuration files, this is the network name of the master which is authorized to send commands to this slave. In all cases (of which I am aware), when you specify a **MASTER** directive, you will also specify **UPSCABLE ether** since your information about the UPS will come via the network from a master.

The slave machine will be shutdown whichever occurs first: either at the request of the master when it does a shutdown or when the values you have specified for TIMEOUT, BATTERYLEVEL, or MINUTES expire (these should work but have not been fully tested). Consequently, if you want the slaves to begin shutting down before the master, you can do so by adjusting the values in the configuration file. If you want the slave to remain up until the master shuts down, you should set TIMEOUT, BATTERYLEVEL, and MINUTES all to zero.

For proper functioning of the slave, you must specify the same **UPSTYPE** in the slave configuration file as is in the master configuration file.

It should be noted that the master and slaves continue to communicate over the network even after the master has issued a shutdown command to the slaves. This is because the master apcupsd continues to run until it receives the shutdown signal from the system. This is important to ensure that all the slaves have been properly notified of the shutdown.

We recommend that the machine names used on the **MASTER** and **SLAVE** directives be put in your /etc/hosts file so that apcupsd will be able to resolve the machine name during startup and shutdown even if DNS is not running. Alternatively, you can use IP addresses on the **MASTER** and **SLAVE** directives, but this is less flexible.

SLAVE <name of slave(s)> used only in MASTER configuration files. Used in master configuration files, this is the name of a slave machine that depends on this master. There can be a maximum of 20 slaves attached to one master. Thus you can specify multiple SLAVE directives in a master configuration file. Only one slave name can be specified per SLAVE directive, thus for multiple slaves, specify multiple SLAVE directives.

As noted above the master and slaves continue to communicate over the network even after the master has issued a shutdown command to the slaves. This is because the master apcupsd continues to run until it receives the shutdown signal from the system. This is important to ensure that all the slaves have been properly notified of the shutdown.

We recommend that the machine names used on the **MASTER** and **SLAVE** directives be put in your /etc/hosts file so that apcupsd will be able to resolve the machine name during startup and shutdown even if DNS is not running. Alternatively, you can use IP addresses on the **MASTER** and **SLAVE** directives, but this is less flexible.

USERMAGIC < user defined magic> used only in SLAVE configuration files. The USERMAGIC directive is a sort of password that gives a second level of identification security in a slave configuration file. It is a character string up to 17 characters in length. It should be unique for each slave. When the slave makes initial contact with the master, this string is passed to the master. Then on each transmission from the master to the slave, the string is passed back to the slave, which checks that it is the correct string before accepting the master's information. This string should be different for each and every slave on the network. This directive is not required.

26.6 Configuration Directives Used to Set the UPS EPROM

NOTE. THESE ARE DEPRECIATED AND NO LONGER WORK IN APCUPSD PLEASE USE APCTEST

The values specified with the following directives are only used if the `--configure` option is specified on the apcupsd command line, and the UPS is capable of internal EPROM programming. In that case, apcupsd attempts to set the values into the UPSes EPROM.

Under normal operations, the values for these parameters specified in the configuration file are not used. Instead, they are read from the UPS EPROM by apcupsd. See Configuration Directives Used to Set the UPS EPROM of this manual for further details before attempting to reprogram your EEPROM.

SENSITIVITY <**sets sensitivity level**> (H)igh, (M)edium, (L)ow

This value determine how sensitive the UPS is to the mains quality and voltage fluctuations. The more sensitive it is, the quicker the UPS will switch to battery power when the mains line quality is bad. Normally, this should be set to H, but if you find your UPS switching to batteries frequently, you might want to try a less sensitive setting, providing that your computer equipment tolerates the poor quality mains. This value is written to the UPS EPROM when the configure option is specified.

Under normal apcupsd operations (no `--configure` option), apcupsd will read the value stored in the UPS and display it in the STATUS output.

WAKEUP <**set wakeup delay**> The UPS power restart delay value in [0,60,180,300] in seconds after the UPS has shut down during a power failure. This is to prevent the power from coming back on too quickly after a power down, and is important for those who have high rpm drives that need to spin down before powering them up again. Some older SCSI models are very sensitive to this problem. Default is zero. This value is written to the UPS EPROM when the `--configure` option is specified.

Under normal apcupsd operations (no `--configure` option), apcupsd will read the value stored in the UPS and display it in the STATUS output.

SLEEP <**set sleep delay**> The UPS delay or grace period in [20,180,300,600] seconds before the UPS cuts the power to your equip-

ment. The default is 20 seconds. This value is written to the UPS EPROM when the `--configure` option is specified.

Under normal apcupsdoperations (no `--configure` option), apcupsd will read the value stored in the UPS and display it in the STATUS output.

LOTRANSFER <lower limit of ups batt. transfer> This sets the low line voltage point at which to switch over to batteries. Different values are permitted based on the UPS model, classification, and manufacture date. Use `apcaccess eeprom` to show you which values are permitted. This value is written to the UPS EPROM when the `--configure` option is specified.

Under normal apcupsdoperations (no `--configure` option), apcupsd will read the value stored in the UPS and display it in the STATUS output.

HITRANSFER <upper limit of ups batt. transfer> This sets the high line voltage point to switch over to batteries. Different values are permitted based on the UPS model, classification, and manufacture date. Use `apcaccess eeprom` to show you which values are permitted. This value is written to the UPS EPROM when the `--configure` option is specified.

Under normal apcupsdoperations (no `--configure` option), apcupsd will read the value stored in the UPS and display it in the STATUS output.

RETURNCHARGE <min. batt. charge level> This parameter specifies what battery percentage charge is necessary before the UPS will supply power to your equipment after a power down. Different values are permitted based on the UPS model, classification, and manufacture date. Use `apcaccess eeprom` to show you which values are permitted. This value is written to the UPS EPROM when the `--configure` option is specified.

Under normal apcupsdoperations (no `--configure` option), apcupsd will read the value stored in the UPS and display it in the STATUS output.

BEEPSTATE <alarm beep state> This parameter tells the UPS when it can sound its audio alarm. These settings are based on discrete events related to the remaining capacity of the UPS.

0 immediately upon power failure

T power failure + 30 seconds

L low battery power

N never

UPSNAME <**string**> This is an eight character string. This is the UPS name that will be stored in the UPS EPROM.

BATTDATE <**string**> This is an eight character string that is the last date the batteries were changed.

27 apcupsd Status Logging

There is a good deal of information available about the UPS and apcupsd's status. This document describes the format of that information. Normally you will get at it via apcaccess, but there are other ways as well.

27.1 Status report format

The STATUS output is in ASCII format with a single data value or piece of information on each line output. Because not all UPSes supply the same information, the output varies based on the type of UPS that you are using. In general, if the information is not available for your UPS, the data portion of the output record will contain an **N/A** indicating that the information is not available.

Status logging consists of periodically logging ALL available information concerning the UPS. Since the volume of data is rather large (over 1000 bytes per status), the STATUS data is not automatically sent to the system log file, instead, it is written as a series of data records to a specific file (normally /etc/apcupsd/apcupsd.status).

After each write, the file is rewound so that the size of the file remains constant. At the current time, this file is 1135 bytes. The format of this file is very similar to the old apcupsd procfs file. The STATUS file is kept for backward compatibility and will be eliminated in a future version of apcupsd. The preferred method for obtaining this information is from apcaccess or by using the CGI interface (see apcupsd Network Monitoring [CGI] Programs).

To make reading the status data reliable via a named pipe, the first record written contains a version number, the number of records that follow the first record, and the total number of bytes in those subsequent records. An actual example of such a status file (/etc/apcupsd/apcupsd.status) is:

Consequently, the first record always consists of 24 bytes (23 characters followed by a newline). This record starts with APC and as indicated in the

example above is followed by 28 records consisting of 675 bytes. The last record begins with END APC and contains the date and time matching the DATE record.

Documentation of each record needs to be written. In the coming weeks, I plan to add additional records and possibly change the names of some of the fields.

When this data is written to a file, it is written as two records, the first record, and all the other records together. In reading the file, it can be either be read a record at a time, or in one big read.

When this data is written to syslog(), it is written a record at a time. The first record is the first 24 bytes. By having the number of records and the size in the first record, the complete status can be reliably reassembled.

27.2 Status Report Example

An example of output from an international SmartUPS 1000 follows:

```
DATE     : Wed Sep 27 17:30:23 CEST 2000
HOSTNAME : polymatou.sibbald.com
RELEASE  : 3.7.3-20000925
CABLE    : Custom Cable Smart
MODEL    : SMART-UPS 1000
UPSMODE  : Stand Alone
STARTTIME: Wed Sep 27 10:39:23 CEST 2000
UPSNAME  : UPS_IDEN
STATUS   : ONLINE
LINEV    : 235.3 Volts
LOADPCT  :   9.3 Percent Load Capacity
BCHARGE  : 100.0 Percent
TIMELEFT : 130.0 Minutes
MBATTCHG : 5 Percent
MINTIMEL : 3 Minutes
MAXTIME  : 0 Seconds
MAXLINEV : 239.2 Volts
MINLINEV : 234.0 Volts
OUTPUTV  : 236.6 Volts
SENSE    : High
DWAKE    : 000 Seconds
DSHUTD   : 020 Seconds
DLOWBATT : 02 Minutes
LOTRANS  : 196.0 Volts
HITRANS  : 253.0 Volts
RETPCT   : 000.0 Percent
ITEMP    : 32.8 C Internal
```

```
ALARMDEL : 5 seconds
BATTV    : 27.9 Volts
LINEFREQ : 50.0 Hz
LASTXFER : Line voltage notch or spike
NUMXFERS : 0
XONBATT  : N/A
TONBATT  : 0 seconds
CUMONBATT: 0 seconds
XOFFBATT : N/A
SELFTEST : NO
STESTI   : 336
STATFLAG : 0x08 Status Flag
DIPSW    : 0x00 Dip Switch
REG1     : 0x00 Register 1
REG2     : 0x00 Register 2
REG3     : 0x00 Register 3
MANDATE  : 07/31/99
SERIALNO : QS9931125245
BATTDATE : 07/31/99
NOMOUTV  : 230
NOMBATTV :  24.0
HUMIDITY : N/A
AMBTEMP  : N/A
EXTBATTS : 0
BADBATTS : N/A
FIRMWARE : 60.11.I
APCMODEL : IWI
END APC  : Wed Sep 27 17:30:31 CEST 2000
```

27.3 Status Report Fields

The meaning of the above variables are:

APC is the header record indicating the STATUS format revision level, the number of records that follow the APC statement, and the number of bytes that follow the record.

DATE is the date and time that the information was last obtained from the UPS.

HOSTNAME is the name of the machine that collected the UPS data.

RELEASE is the apcupsd release number.

CABLE is the cable as specified in the configuration file.

MODEL is the UPS model as derived from information from the UPS.

UPSMODE is the mode in which apcupsd is operating.

STARTTIME is the time/date that apcupsd was started.

UPSNAME is the name of the UPS as stored in the EEPROM.

STATUS is the current status of the UPS (ONLINE, CHARGING, ONBATT,...)

MASTERUPD is the last time the master sent an update to the slave. This value is present only in slave configurations.

LINEV is the current line voltage as returned by the UPS.

LOADPCT is the percentage of load capacity as estimated by the UPS.

BCHARGE is the percentage charge on the batteries.

TIMELEFT is the remaining runtime left on batteries as estimated by the UPS.

MBATTCHG if the battery charge percentage (BCHARGE) drops below this value, apcupsd will shutdown your system.

MINTIMEL apcupsd will shutdown your system if the remaining runtime equals or is below this point.

MAXTIME apcupsd will shutdown your system if the time on batteries exceeds this value. A value of zero disables the feature.

MAXLINEV is the maximum line voltage since the last STATUS as returned by the UPS.

MINLINEV is the minimum line voltage since the last STATUS as returned by the UPS.

OUTPUTV is the voltage the UPS is supplying to your equipment.

SENSE is the sensitivity level of the UPS to line voltage fluctuations.

DWAKE is the amount of time the UPS will wait after a power off condition when the power is restored.

DSHUTD is the grace delay that the UPS gives after receiving a power down command from apcupsd before it powers off your equipment.

DLOWBATT is the remaining runtime below which the UPS sends the low battery signal. At this point apcupsd will force an immediate emergency shutdown.

LOTRANS is the line voltage below which the UPS will switch to batteries.

HITRANS is the line voltage above which the UPS will switch to batteries.

RETPCT is the percentage charge that the batteries must have after a power off condition before the UPS will restore power to your equipment.

STATFLAG is a status flag indicating the UPS status. See STATUS.

ITEMP is the internal UPS temperature as supplied by the UPS.

ALARMDEL is the delay period for the UPS alarm.

BATTV is the battery voltage as supplied by the UPS.

LINEFREQ is the line frequency in Hertz as given by the UPS.

LASTXFER is the reason for the last transfer to batteries.

NUMXFERS the number of transfers to batteries since apcupsd startup.

XONBATT time and date of last transfer to batteries, or N/A.

TONBATT time in seconds currently on batteries, or 0.

CUMONBATT total (cumulative) time on batteries in seconds since apcupsd startup.

XOFFBATT time and date of last transfer from batteries, or N/A.

SELFTEST is the results of the last self test, and may have the following values: OK - self test indicates good battery BT - self test failed due to insufficient battery capacity NG - self test failed due to overload NO - No results (i.e. no self test performed in the last 5 minutes).

STESTI is the interval in hours between automatic self tests.

STATFLAG status flag. English version is given by STATUS.

DIPSW is the dip switch settings.

REG1 is the value from the UPS fault register 1.

REG2 is the value from the UPS fault register 2.

REG3 is the value from the UPS fault register 3.

MANDATE is the date the UPS was manufactured.

SERIALNO is the UPS serial number.

BATTDATE is the date that batteries were last replaced.

NOMOUTV is the output voltage that the UPS will attempt to supply when on battery power.

NOMBATTV is the nominal battery voltage.

HUMIDITY is the humidity as measured by the UPS.

AMBTEMP is the ambient temperature as measured by the UPS.

EXTBATTS is the number of external batteries as defined by the user. A correct number here helps the UPS compute the remaining runtime more accurately.

BADBATTS is the number of bad battery packs.

FIRMWARE is the firmware revision number.

APCMODEL is the old APC model identification code.

END APC is the time and date that the STATUS record was written.

27.4 Logging the STATUS Information

If specified in the configuration file, the STATUS data will also be written to the system log file. Please note, that it would not normally be wise to write this data to a normal system log file as there is no mechanism in syslog() to rewind the file and hence the log file would quickly become enormous. However, in two cases, it can be very useful to use syslog() to write this information.

The first case is to set up your syslog.conf file so that the data is written to a named pipe. In this case, normally not more than about 8192 bytes of data will be kept before it is discarded by the system.

The second case is to setup your syslog.conf file so that the status data is sent to another machine, which presumably then writes it to a named pipe. Consequently, with this mechanism, provides a simple means of networking apcupsd STATUS information.

Although we mention system logging of STATUS information, we strongly recommend that you use apcaccess or the CGI interface to get this information.

28 The Shutown Sequence and its Discontents

28.1 Shutdown Sequence

If you experienced so problems with the testing procedures, or if you are porting apcupsd to another system, or you are simply curious, you may want to know exactly what is going on during the shutdown process.

The shutdown sequence is as follows:

- apcupsd detects that there is a power problem and it calls `/etc/apcupsd/apccontrol powerout`, which normally sends a message to all users informing them of a potential problem.
- After approximately 5 seconds in the power problem mode, apcupsd calls `/etc/apcupsd/apccontrol onbattery`, which normally sends a message to all users informing them that the UPS is on batteries.
- When one of the conditions listed below occurs, apcupsd issues a shutdown command by calling `/etc/apcupsd/apccontrol doshutdown`, which should perform a shutdown of your system using the system shutdown(8) command. You can modify the behavior by editing the /etc/apcupsd/apccontrol script, but doing so will make it more complicated to upgrade to the next apcupsd version.

  The conditions that trigger the shutdown can be: running time on batteries have expired (TIMEOUT), the battery runtime remaining is below the configured value (BATTERYLEVEL), the estimated remaining runtime is below the configured value (MINUTES), or the UPS signals that the batteries are exhausted.

  A shutdown could also be initiated if apcupsd detects that the batteries are no longer functioning correctly. This case, though very unusual, can happen at any time even if there is proper mains voltage, and `/etc/apcupsd/apccontrol emergency` is called.

  Just before initiating any shutdown through the apccontrol script, apcupsd will create the file /etc/apcupsd/powerfail. This file will be used later in the shutdown sequence to recall apcupsd after syncing of the disks to initiate a power off of the UPS.

  If the /etc/nologin file has not already been created, it will normally be created during the shutdown sequence to prevent additional users from logging in (see the NOLOGIN configuration directive).

  Even though apcupsd has requested the system to perform a shutdown, it continues running. If it is a master with slaves, it will inform the

slaves to do a shutdown. They perform their shutdown by calling `/etc/apcupsd/apccontrol remotedown`.

- When the system signals apcupsd to do exit, it does so. This is part of the normal system shutdown (at least on Unix and Linux systems) and the exact time that apcupsd receives the termination signal depends on how the shutdown links (usually in /etc/rc.d) are set.

  Note that on Windows NT systems, apcupsd apparently continues to run as a Service even though the machine is "shutdown".

- During the shutdown of the system after apcupsd has been forced to exit, one of the last things done by the system shutdown is to call the halt script, which is usually in /etc/rc.d/halt or /etc/rc.d/init.d/halt, or possibly in /sbin/init.d/rc.0 depending on your system. If apcupsd was properly installed, this standard halt script was modified to include a bit of new logic just before the final halt of the system. It first tests if the file /etc/apcupsd/powerfail exists, and if it does, it executes `/etc/apcupsd/apccontrol killpower`. It is this last step that will cause apcupsd to be re-executed with the `--killpower` option on the command line. This option tells apcupsd to inform the UPS to kill the power.

  This final step is important if you want to ensure that your system will automatically reboot when the power comes back on. The actual code used on the Red Hat version is:

```
# See if this is a powerfail situation.                                  # ***apcupsd***
if [ -f /etc/apcupsd/powerfail ]; then                                   # ***apcupsd***
 echo                                                                    # ***apcupsd***
 echo "APCUPSD will now power off the UPS"                               # ***apcupsd***
 echo                                                                    # ***apcupsd***
 /etc/apcupsd/apccontrol killpower                                       # ***apcupsd***
 echo                                                                    # ***apcupsd***
 echo "Please ensure that the UPS has powered off before rebooting"      # ***apcupsd***
 echo "Otherwise, the UPS may cut the power during the reboot!!!"        # ***apcupsd***
 echo                                                                    # ***apcupsd***
fi                                                                       # ***apcupsd***
```

The above code must be inserted as late as possible in the halt script. On many systems, such as Red Hat, all the disk drives were unmounted, then remounted read-only, thus permitting access to the /etc files and the apcupsd executable. If your system does not explicitly remount the disks, you must remount them in read-only mode in the code that you add. Examples of code fragments that do this can be found in the distributions/suse subdirectory of the source.

If you are not able to insert the above code in your halt script because there is no halt script, or because your halt script calls the init program as some Unix systems do, you can either just forget about powering off the UPS, which means that your machine will not automatically reboot after a power failure, or there is yet another alternative, though not at all as satisfying as inserting code in the halt script.

Only if you cannot insert the appropriate code in the halt script, when you start apcupsd, normally from the /etc/rc.d/init.d/apcupsd script, use the `--kill-on-powerfail` option. This will cause apcupsd to program the UPS to shutoff the power just before it (apcupsd) does the system shutdown. Please note that this is not the most ideal solution. Read on to understand why.

A very important consideration is that you must set the EEPROM in your UPS so that it waits a sufficient time for the system to halt before it shuts off the UPS power. The current value as well as the permitted values for your UPS can be determined by executing:

```
apcaccess eeprom
```

The output should look something like the following:

```
apcaccess eeprom

Valid EPROM values for the SMART-UPS 1000

                       Config        Current  Permitted
Description              Directive     Value    Values
=================================================================
Upper transfer voltage   HITRANSFER    253      253 264 271 280
Lower transfer voltage   LOTRANSFER    196      196 188 208 204
Return threshold         RETURNCHARGE  0        00 15 50 90
Output voltage on batts  OUTPUTVOLTS   230      230 240 220 225
Sensitivity              SENSITIVITY   H        H M L L
Low battery warning      LOWBATT       2        02 05 07 10
Shutdown grace delay     SLEEP         20       020 180 300 600
Alarm delay              BEEPSTATE     0        0 T L N
Wakeup delay             WAKEUP        0        000 060 180 300
Self test interval       SELFTEST      336      336 168 ON  OFF
```

The line of interest for you is the **Shutdown grace delay**, which can be changed using the SLEEP (see SLEEP <set sleep delay>) directive in your

apcupsd.conf file. The default value is 20 seconds, but generally, you can set it to 180, 300, or 600 seconds depending on your UPS. See the EEPROM (see Configuring Your EEPROM) this manual for further details on how to change this EPROM value.

If you use the `--kill-on-powerfail` option, you run the risk of having the computer power cut before the system has shutdown. Even if the grace period is rather long, if something goes wrong in the shutdown, well, it is up to you to decide.

If apcupsd has successfully shutdown your computer and powered off the UPS during a power outage, you can control whether or not your computer is automatically rebooted when the power returns.

The UPS contains two internal EPROM values that determine when it will restore power to your computer after a full power shutdown. They are the RETURNCHARGE (see RETURNCHARGE <min_ batt_ charge level>) percentage and the WAKEUP (see WAKEUP <set wakeup delay>) delay. Briefly, the **RETURNCHARGE** specifies what percentage charge the battery must have before the power is restored. Higher values are recommended in regions where the power goes up and down frequently. The **WAKEUP** delay is a simple time delay. Most sites will have both of these at zero, or perhaps the **RETURNCHARGE** set to 15. Please follow the links to the **Configuration** section of this manual for more information. See the EEPROM (see Configuring Your EEPROM) of this manual for further details on how to change these EPROM values.

28.2 Shutdown Problems

Obviously if your halt script is not properly modified, apcupsd will not be able to shut off the power to the UPS, and if the power returns before the batteries are exhausted your system will not automatically reboot. In any case, your machine should have been cleanly shut down.

28.3 Master/Slave Shutdown

In master/slave configurations, however, the master cannot be 100 percent sure that the slaves have all shutdown before it performs the power off. As a consequence, it is possible that the master will shut off the power before the slave has finished shutdown. If this is the case, the best procedure is to put an appropriate sleep command in the /etc/apcupsd/apccontrol file on the master. For example to give the slaves 30 additional seconds to shutdown,

one would add:

```
sleep 30
```

just after the line that reads

```
doshutdown)
```

in the apccontrol file (approximately line 79 – depending on your system version).

Also, on a slave machine, you do not want to use the modified halt script since it will recall apcupsd, which will detect that it is a slave (i.e. no connection to the UPS) and will complain that it cannot do the killpower. This situation is not harmful just annoying and possibly confusing.

One possible problem during shutdown can be caused by remnants of old versions. Please be sure to delete or rename all prior versions (/usr/local/sbin/apcupsd or /sbin/powersc).

28.4 Startup

Normally, apcupsd is automatically started when your system is rebooted. This normally occurs because the startup script apcupsd is linked into the appropriate places in /etc/rc.d. On most Linux systems, there is a program called chkconfig(8) that will automatically link the startup script. This program is invoked by the `make install` scripts, or it is explicitly done for those systems that do not have chkconfig(8). If this is not the case, you can either link it in appropriately yourself or explicitly call it from your rc.local file. The appropriate manual way to startup apcupsd is by executing:

```
<path>/apcupsd start
```

where <path> is normally /etc/rc.d or /etc/rc.d/init.d depending on your system (isn't Unix wonderful? :-)). Using this script is important so that any files remaining around after a power failure are removed. Likewise, shutting down apcupsd should be done with the same script:

```
<path>/apcupsd stop
```

28.5 Windows Considerations

Please see the end of Windows chapter (see Installation on Windows) of this manual for conderations pertaining to shutdown and killpower on Windows.

29 APC smart protocol

The APC UPS protocol was originally analyzed by Pavel Korensky with additions from Andre H. Hendrick beginning in 1995, and we want to give credit for good, hard work, where credit is due. After having said that, you will see that Steven Freed built much of the orginal apcupsd information file. [Comment inserted by Riccardo Facchetti]

The start of this chapter of the apcupsd manual in HTML format was pulled from the Network UPS Tools (NUT) site. It has been an invaluable tool in improving apcupsd, and I consider it the **Bible** of APC UPS programming. In the course of using it, I have added information gleaned from apcupsd and information graciously supplied by APC. Hopefully, the additions made herein can benefit the original author and his programming project, and maybe some day, the apcupsd project and the **NUT** project can join forces.

29.1 Description

Here's the information on the elusive APC smart signaling protocol used by their higher end units (Back-UPS Pro, Smart-UPS, Matrix-UPS, etc). What you see here has been collected from a variety of sources. Some people analyzed the chatter between PowerChute and their hardware. Others sent various characters to the UPS and figured out what the results meant.

29.2 RS-232 differences

Normal 9 pin serial connections have TxD on 3 and RxD on 2. APC's smart serial ports put TxD on pin 1 and RxD on pin 2. This means you go nowhere if you use a normal straight through serial cable. In fact, you might even power down the load if you plug one of those cables in. This is due to the odd routing of pins - DTR and RTS from the PC usually wind up driving the on/off line. So, when you open the port, they go high and *poof* your computer dies.

Originally this evil hack was used to connect the UPS to the PC when this page was first being built. As you can see, I cheated and neglected the

ground (only 2 wires!) and it still worked. This method can be used for playing around, but for professional systems this is obviously not a viable option.

That hack didn't work out so well (damned cats), so it was retired quite awhile back. The most practical solution was to go out and BUY the DOS/Win version of PowerChute just for the black (smart) cable. I recommend doing the same thing if you actually care about this thing working properly. Of course, if you have one of the newer packages that came with PowerChute, you already have the cable you need.

29.3 Diagram for cable hackers

If you are handy with cable creation tools, check out the 940-0024C clone diagram. That's the black "smart" cable normally provided with APC models sold after 1996. The loopback pins on that diagram are used to keep PowerChute happy by allowing cable detection. If you use the NUT apcsmart driver, those pins don't matter.

Many thanks to Steve Draper for providing this scan.

For additional information on cables, see the section on custom cables (see Cables) in this manual.

29.4 The Smart Protocol

Despite the lack of official information from APC, this table has been constructed. It's standard RS-232 serial communications at 2400 bps/8N1. Don't rush the UPS while transmitting or it may stop talking to you. This isn't a problem with the normal single character queries, but it really does matter for multi-char things like "@000". Sprinkle a few calls to usleep() in your code and everything will work a lot better.

The following table describes the single character **Code** or command that you can send to the UPS, its meaning, and what sort of response the UPS will provide. Typically, the response shown below is followed by a newline (\n in C) and a carriage return (\r in C). If you send the UPS a command that it does not recognize or that is not available on your UPS, it will normally respond by "NA" for not available, otherwise the response is given in the "Typical results" column. >

Code	Meaning	Typical results
^A	Model string	SMART-UPS 700

^N	Turn on UPS (send twice, with > 1.5s delay between chars) Only on 3rd gen SmartUPS and Black Back-UPS Pros	n/a
^Z	Permitted EEPROM Values	A large string (254 chars) that gives the EEPROM permitted values for your model. For details see below.
A	Front panel test	Light show + "OK" (and 2s beep)
B	Battery voltage	Ranges - typical "27.87"
C	Internal temperature (degrees C)	Ranges - typical "036.0"
D	Runtime calibration - runs until battery is below 25% (35% for Matrix) This updates the 'j' values - only works at 100% battery charge. Can be aborted with a second "D"	! when on battery, $ on line
E	Automatic self test intervals	Default = 336 (336 hours = 14 days) (336=14 days, 168=7 days, ON=power on, OFF=never)
F	Line frequency, Hz	60.00 (50.0 in Europe)
G	Cause of transfer	R = unacceptable utility voltage rate of change, H = high utility voltage, L = low utility voltage, T = line voltage notch or spike, O = no transfers yet (since turnon), S = transfer due to serial port U command or activation of UPS test from front panel, NA = transfer reason still not available (read again).
K—K	Shutdown with grace period (set with 'p') - need > 1.5s between first and second K	Matrix/3rd gen SmartUPS/Black Back-UPS Pros: "OK", all others: "*"
L	Input line voltage	Ranges - typical "118.3" or "228.8" in Europe
M	Maximum line voltage received since last M query	Ranges - typical "118.9" or "230.1" in Europe
N	Minimum line voltage received since last N query	Ranges - typical "118.9" or "226.2" in Europe
O	Output voltage	Ranges - typical "118.3" or "228.8" in Europe
P	Power load %	Ranges - typical "011.4" depends on what you have plugged in.

Q	Status flags	Bitmapped, see below
R	Turn dumb Only on 3rd gen SmartUPS, SmartUPS v/s, BackUPS Pro	“BYE”
S	Soft shutdown after ’p’ delay, return online when power returns Only works when UPS is on battery	OK
U	Simulate power failure	!! when switching to battery, then $ when back on line
V	Old firmware revision	“GWD” or “IWI” The last character indicates the locale (Domestic, International).
W	Self test (battery), results stored in “X”	“OK”
X	Results of last self test	“OK” - good battery, “BT” - failed due to insufficient capacity, “NG” - failed due to overload, “NO” - no results available (no test performed in last 5 minutes)
Y	Enter smart mode	“SM”
Z—Z	Shutdown immediately (no delay) - need > 1.5s between first and second Z	N/A
a	Show protocol version.alert messages.valid commands (delimited by periods)	“3.!$%+?=#—.^A^N^Z+-789<@ABCDEFGKLMNOPQRSUVWXYZ’abcefgjklm - Link-Level.alert-messages.commands
b	Firmware revision	“50.9.D” - 50 = SKU (variable length), 9 = firmware revision, D = country code (D=USA, I=International, A=Asia, J=Japan, M=Canada)
c	UPS local id	UPS_IDEN (you can program any 8 characters here)
e	Return threshold	% battery charge threshold for return (00=00%, 01=15%, 02=25%, 03=90%)
f	Battery level %	Ranges - typical “100.0” when fully charged as should normally be the case
g	Nominal battery voltage (not actual voltage - see B)	“012” or “024” or “048”.

h	Measure-UPS: ambient humidity (%)	"nnn.n" - percentage
i	Measure-UPS: dry contacts	10 = contact 1, 20 = 2, 40 = 3, 80 = 4
j	Estimated runtime at current load (minutes)	"0112:" (note, it is terminated with a colon)
k	Alarm delay	0(zero) = 5 second delay after fail, T = 30 second delay, L = alarm at low battery only, N = no alarm
l	Low transfer voltage	Default "103" or "208" in Europe
m	Manufacturing date	Unique within groups of UPSes (production runs)
n	Serial number	Unique for each UPS
o	Nominal Output Voltage	The Nominal Output Voltage when running on batteries. Default "115" or "230" in Europe.
p	Shutdown grace delay, seconds	Default "020" (020/180/300/600)
q	Low battery warning, minutes	Default "02"
r	Wakeup delay (time) - seconds	Default "000" (000/060/180/300)
s	Sensitivity	"H" - highest, "M" - medium, "L" - lowest, "A" - autoadjust (Matrix only)
u	Upper transfer voltage	Default "132" or "253" in Europe
t	Measure-UPS: ambient temperature (degrees C)	"nn.nn"
x	Last battery change	Eight characters. Varies typically dd/mm/yy - 31/12/99
y	Copyright notice	"(C) APCC" - only works if firmware letter (from "V") is later than O
z	Reset the EEPROM to factory settings (but not ident or batt replacement date) Not on SmartUPS v/s or BackUPS Pro	"CLEAR"
+	Capability cycle	Cycle forward through possible values ("—" from UPS afterward to confirm change). Do not use this unless you know how to program your UPS EEPROM or you may damage your UPS.

-	Capability cycle	Cycle backward through possible values (“—” from UPS afterward to confirm change)Do not use this unless you know how to program your UPS EEPROM or you may damage your UPS.
@nnn	Shutdown (after delay ’p’) with delayed wakeup of nnn tenths of an hour (after ’r’ time)	Matrix/3rd gen UPS: “OK”, others “*”
0x7f (DEL key)	Abort shutdown - use to abort @, S, K—K	“OK”
~	Register #1	See below
’	Register #2	See below
0	Battery constant	Set to A0 on SmartUPS 1000 with new battery
4	???	Prints 35 on SmartUPS 1000
5	???	Prints EF on SmartUPS 1000
6	???	Prints F9 on SmartUPS 1000
7	Dip switch positions (if applicable)	See below
8	Register #3	See below
9	Line quality	“FF” acceptable, “00” unacceptable
>	Number of external battery packs attached	SmartCell models: “nnn” where nnn is how many external packs are connected Non-SmartCell units: whatever has been set with >+ and >- by the user
Matrix UPS (and possibly Symmetra) specific commands		
^	Run in bypass mode	If online, “BYP” is received as bypass mode starts If already in bypass, “INV” is received and UPS goes online “ERR” received if UPS is unable to transfer

<	Number of bad battery packs	“nnn” - count of bad packs connected to the UPS
/	Load current	“nn.nn” - true RMS load current drawn by UPS
\	Apparent load power	“nnn.nn” - output load as percentage of full rated load in VA.
^V	Output voltage selection (editable)	“A” - automatic according to input tap, “M” - 208 VAC, “I” - 240 VAC
^L	Front panel language	“E” - English, “F” - French, “G” - German, “S” - Spanish, “1” “2” “3” “4” - ?
w	Run time conservation	“NO” (disabled) or “02” “05” “08” - minutes of runtime to leave in battery (UPS shuts down “early”)

29.5 Dip switch info

Bit	Switch	Option when bit=1
0	4	Low battery alarm changed from 2 to 5 mins. Autostartup disabled on SU370ci and 400
1	3	Audible alarm delayed 30 seconds
2	2	Output transfer set to 115 VAC (from 120 VAC) or to 240 VAC (from 230 VAC)
3	1	UPS desensitized - input voltage range expanded
4-7	-	Unused at this time

29.6 Status bits

This is probably the most important register of the UPS, which indicates the overall UPS status. Some common things you’ll see:

- 08 = On line, battery OK
- 10 = On battery, battery OK
- 50 = On battery, battery low
- SM = Status bit is still not available (retry reading)

Bit	Hex Bit	Meaning
0	0x01	1 = Runtime calibration occurring Not reported by Smart UPS v/s and BackUPS Pro
1	0x02	1 = SmartTrim Not reported by 1st and 2nd generation SmartUPS models
2	0x04	1 = SmartBoost
3	0x08	1 = On line (this is the normal condition)
4	0x10	1 = On battery
5	0x20	1 = Overloaded output
6	0x40	1 = Battery low
7	0x80	1 = Replace battery

29.7 Alert messages

These single character messages are sent by the UPS any time there is an Alert condition. All other responses indicated above are sent by the UPS only in response to a query or action command.

Character	Description
!	Line Fail - sent when the UPS goes on-battery, repeated every 30 seconds until low battery condition reached. Sometimes occurs more than once in the first 30 seconds.
$	Return from line fail - UPS back on line power, only sent if a ! has been sent.
%	Low battery - Sent to indicate low battery, but not on SmartUPS v/s or BackUPS Pro models
+	Return from low battery - Sent when the battery has been recharged to some level only if a % has been sent previously
?	Abnormal condition - sent for conditions such as “shutdown due to overload” or “shutdown due to low battery capacity”. Also occurs within 10 minutes of turnon.
=	Return from abnormal condition - Sent when the UPS returns from an abnormal condition where ? was sent, but not a turn-on. Not implemented on SmartUPS v/s or BackUPS Pro models.
*	About to turn off - Sent when the UPS is about to switch off the load. No commands are processed after this character is sent. Not implemented on SmartUPS v/s, BackUPS Pro, or 3rd generation SmartUPS models.

#	Replace battery - Sent when the UPS detects that the battery needs to be replaced. Sent every 5 hours until a new battery test is run or the UPS is shut off. Not implemented on SmartUPS v/s or BackUPS Pro models.
&	Check alarm register for fault (Measure-UPS) - sent to signal that temp or humidity out of set limits. Also sent when one of the contact closures changes states. Sent every 2 minutes, stops when the alarm conditions are reset. Only sent for alarms enabled with I. Cause of alarm may be determined with J. Not on SmartUPS v/s or BackUPS Pro.
—	Variable change in EEPROM - Sent whenever any EEPROM variable is changed. Only supported on Matrix UPS and 3rd generation SmartUPS models.

29.8 Register 1

All bits are valid on the Matrix UPS. SmartUPS models only support bits 6 and 7. Other models do not respond.

Bit	Hex Bit	Meaning
0	0x01	In wakeup mode (typically lasts < 2s)
1	0x02	In bypass mode due to internal fault - see register 2 or 3
2	0x04	Going to bypass mode due to command
3	0x08	In bypass mode due to command
4	0x10	Returning from bypass mode
5	0x20	In bypass mode due to manual bypass control
6	0x40	Ready to power load on user command
7	0x80	Ready to power load on user command or return of line power

29.9 Register 2

Matrix UPS models report bits 0-5. SmartUPS models only support bits 4 and 6. SmartUPS v/s and BackUPS Pro report bits 4, 6, 7. Unused bits are set to 0. Other models do not respond.

Bit	Meaning
0	Fan failure in electronics, UPS in bypass

1	Fan failure in isolation unit
2	Bypass supply failure
3	Output voltage select failure, UPS in bypass
4	DC imbalance, UPS in bypass
5	Command sent to stop bypass with no battery connected - UPS still in bypass
6	Relay fault in SmartTrim or SmartBoost
7	Bad output voltage

29.10 Register 3

All bits are valid on the Matrix UPS and 3rd generation SmartUPS models. SmartUPS v/s and BackUPS Pro models report bits 0-5. All others report 0-4. State change of bits 1,2,5,6,7 are reported asynchronously with ? and = messages.

Bit	Meaning
0	Output unpowered due to shutdown by low battery
1	Unable to transfer to battery due to overload
2	Main relay malfunction - UPS turned off
3	In sleep mode from @ (maybe others)
4	In shutdown mode from S
5	Battery charger failure
6	Bypass relay malfunction
7	Normal operating temperature exceeded

29.11 Interpretation of the Old Firmware Revision

The Old Firmware Revision is obtained with the "V" command, which gives a typical response such as "GWD" or "IWI", and can be interpreted as follows:

```
Old Firmware revision and model ID String for SmartUPS \& MatrixUPS

This is a three character string XYZ

   where X == Smart-UPS or Matrix-UPS ID Code.
     range 0-9 and A-P
```

```
    1 == unknown
    0 == Matrix 3000
    5 == Matrix 5000
  the rest are Smart-UPS and Smart-UPS-XL
    2 == 250       3 == 400       4 == 400
    6 == 600       7 == 900       8 == 1250
    9 == 2000      A == 1400      B == 1000
    C == 650       D == 420       E == 280
    F == 450       G == 700       H == 700XL
    I == 1000      J == 1000XL    K == 1400
    L == 1400XL    M == 2200      N == 2200XL
    O == 3000      P == 5000

 where Y == Possible Level of Smart Features, unknown???
    G == Stand Alone
    T == Stand Alone
            V == ???
    W == Rack Mount

 where Z == National Model Use Only Codes
    D == Domestic        115 Volts
    I == International   230 Volts
    A == Asia ??         100 Volts
    J == Japan ??        100 Volts
```

29.12 Interpretation of the New Firmware Revision

```
New Firmware revison and model ID String in NN.M.L is the format

  where NN == UPS ID Code.
      12 == Back-UPS Pro 650
      13 == Back-UPS Pro 1000
      52 == Smart-UPS 700
      60 == SmartUPS 1000
      72 == Smart-UPS 1400

      where NN now Nn has possible meanings.
          N  == Class of UPS
          1n == Back-UPS Pro
          5n == Smart-UPS
          7n == Smart-UPS NET

           n == Level of intelligence
          N1 == Simple Signal, if detectable WAG(*)
          N2 == Full Set of Smart Signals
          N3 == Micro Subset of Smart Signals

  where M == Possible Level of Smart Features, unknown???
      1 == Stand Alone
      8 == Rack Mount
```

```
    9 == Rack Mount

where L == National Model Use Only Codes
    D == Domestic        115 Volts
    I == International   230 Volts
    A == Asia ??         100 Volts
    J == Japan ??        100 Volts
    M == North America   208 Volts (Servers)
```

29.13 EEPROM Values

Upon sending a ^Z, your UPS will probably spit back approximately 254 characters something like the following (truncated here for the example):

#uD43132135138129uM43229234239224uA43110112114108 ....

It looks bizarre and ugly, but is easily parsed. The # is some kind of marker/ident character. Skip it. The rest fits this form:

- Command character - use this to select the value
- Locale - use 'b' to find out what yours is (the last character), '4' applies to all
- Number of choices - '4' means there are 4 possibilities coming up
- Choice length - '3' means they are all 3 chars long

Matrix-UPS models have ## between each grouping for some reason.

Here is an example broken out to be more readable:

```
CMD DFO RSP FSZ FVL
u   D   4   3   127 130 133 136
u   M   4   3   229 234 239 224
u   A   4   3   108 110 112 114
u   I   4   3   253 257 261 265
l   D   4   3   106 103 100 097
l   M   4   3   177 172 168 182
l   A   4   3   092 090 088 086
l   I   4   3   208 204 200 196
e   4   4   2   00   15  50  90
o   D   1   3   115
o   J   1   3   100
o   I   1   3   230 240 220 225
```

```
o   M   1   3   208
s   4   4   1     H   M   L   L
q   4   4   2    02  05  07  10
p   4   4   3   020 180 300 600
k   4   4   1     0   T   L   N
r   4   4   3   000 060 180 300
E   4   4   3   336 168  ON OFF

CMD == UPSlink Command.
      u = upper transfer voltage
      l = lower transfer voltage
      e = return threshold
      o = output voltage
      s = sensitivity
      p = shutdown grace delay
      q = low battery warning
      k = alarm delay
      r = wakeup delay
      E = self test interval

DFO == (4)-all-countries (D)omestic (I)nternational (A)sia (J)apan
       (M) North America - servers.
RSP == Total number possible answers returned by a given CMD.
FSZ == Max. number of field positions to be filled.
FVL == Values that are returned and legal.
```

29.14 Programming the UPS EEPROM

There are at this time a maximum of 12 different values that can be programmed into the UPS EEPROM. They are:

Item	Command	Meaning
1.	c	The UPS Id or name
2.	x	The last date the batteries were replaced
3.	u	The Upper Transfer Voltage
4.	l	The Lower Transfer Voltage
5.	e	The Return Battery Charge Percentage
6.	o	The Output Voltage when on Batteries
7.	s	The Sensitivity to Line Quality
8.	p	The Shutdown Grace Delay
9.	q	The Low Battery Warning Delay
10.	k	The Alarm Delay
11.	r	The Wakeup Delay
12.	E	The Automatic Self Test Interval

The first two cases (Ident and Batt date) are somewhat special in that you tell the UPS you want to change the value, then you supply 8 characters that are saved in the EEPROM. The last ten item are programmed by telling the UPS that you want it to cycle to the next permitted value.

In each case, you indicate to the UPS that you want to change the EEPROM by first sending the appropriate query command (e.g. "c" for the UPS ID or "u" for the Upper Transfer voltage. This command is then immediately followed by the cycle EEPROM command or "-". In the case of the UPS Id or the battery date, you follow the cycle command by the eight characters that you want to put in the EEPROM. In the case of the other ten items, there is nothing more to enter.

The UPS will respond by "OK" and approximately 5 seconds later by a vertical bar (—) to indicate that the EEPROM was changed.

29.15 Acknowledgements

The apcupsd has a rather long and tormented history. Many thanks to the guys that, with time, contributed to the general public knowledge.

Pavel Korensky <pavelk at dator3.anet.cz>, Andre M. Hedrick <hedrick at suse.de>, Christopher J. Reimer <reimer at doe.carleton.ca>, Kevin D. Smolkowski <kevins at trigger.oslc.org>, Werner Panocha <wpanocha at t-online.de>, Steven Freed, Russell Kroll.

additions by: Kern Sibbald <apcupsd-users at lists.sourceforge.net>

30 Apcupsd — RPM Packaging FAQ

30.1 Answers

How do I build Apcupsd for platform xxx? The apcupsd spec file contains defines to build for several platforms: RedHat 7.x (rh7), RedHat 8.0 (rh8), RedHat 9 (rh9), Fedora Core 1 (fc1), and Whitebox Enterprise Linux 3.0 (wb3). The package build is controlled by a define set at the beginning of the file. These defines basically just control the dependancy information that gets coded into the finished rpm package. So while you could technically build a package without defining a platform, or with an incorrect platform, and have it install and run it would not contain correct dependancy information for the rpm database. The platform define may be edited in the spec file di-

rectly (by default all defines are set to 0 or "not set"). For example, to build the RedHat 7.x package find the line in the spec file which reads

```
%define rh7 0
```

and edit it to read

```
%define rh7 1
```

Alternately you may pass the define on the command line when calling rpmbuild:

```
rpmbuild -ba --define "build_rh7 1" apcupsd.spec
rpmbuild --rebuild --define build_rh7 1" apcupsd-x.x.x-x.src.rpm
```

How do I control whether usb support gets built? By default standard serial port support will be built and the apcupsd-std package will be produced. To get the usb package and support either set the

```
%define usb 0
```

to

```
%define usb 1
```

in the spec file directly or pass it to rpmbuild on the command line:

```
rpmbuild -ba --define "build_rh7 1" --define "build_usb 1" apcupsd.spec
```

other defines are used? The only other define is for the initdir for the daemon control script. On RedHat systems this is set to /etc/rc.d/init.d/. You would only need to edit this if packaging for a platform with uses a different directory.

I'm getting errors about not having permission when I try to build the packages. Do I need to be root? No, you do not need to be root and, in fact, it is better practice to build rpm packages as a non-root user. Apcupds packages are designed to be built by a regular user but you must make a few changes on your system to do this. If you are building on your own system then the simplest method is to add write permissions for all to the build directory (/usr/src/redhat/). To accomplish this execute the following command as root:

```
chmod -R 777 /usr/src/redhat
```

If you are working on a shared system where you can not use the method above then you need to recreate the /usr/src/redhat directory tree with all of it's subdirectories inside your home directory. Then create a file named

```
.rpmmacros
```

in your home directory (or edit the file if it already exists) and add the following line:

```
%_topdir /home/myuser/redhat
```

=“appendix”¿Appendix A Credits

The success of apcupsd is due to the many people that helped in development, testing and in many other ways.

Thank all the developers that worked hard to make APCUPSDone of the best piece of software for UPSmanagement.

A.1 Contributors

Current Code Maintainer and Project Manager: Kern Sibbald (kern at sibbald dot com)

RPM Packager: D. Scott Barninger (barninger at fairfieldcomputers dot com)

Super USB bug fixer: Adam Kropelin (akropel1 at rochester dot rr dot com)

CGI and HTML fixer: William King (wrking at dadaboom dot com)

Project Starter and Former Code Maintainer: Andre Hedrick (andre at linux-ide.org)

Former Code Maintainer and Project Manager: Riccardo Facchetti (riccardo at master.oasi.gpa.it)

Serial Communications: Andre Hedrick (andre at linux-ide.org)

2.0 User's Manual: Eric S. Raymond (esr at thyrsus.com)

Alpha Port: Kern Sibbald (kern at sibbald dot com)

J. Rochate (jrochate at ualg.pt) testing and machine loan

Caldera: John Pinner (john at clocksoft.com)

HP-UX Port Carl Erhorn (Carl_Erhorn at hyperion.com)

Robert K Nelson (rnelson at airflowsciences.com)

SOLARIS Port: Carl Erhorn (Carl_Erhorn at hyperion.com)

OpenBSD Port: Devin Reade (gdr at gno.org)

NetBSD Port: Neil Darlow (neil at darlow.co.uk)

Win32 Port: Kern Sibbald (kern at sibbald dot com)

Paul Z. Stagner (paul.stagner at charterco.com) testing

WEB Interfaces: Kern Sibbald (kern at sibbald dot com)

Joseph Acosta (joeja at mindspring.com)

Apcupsd Support and Knowledge Base: Brian Schau (Brian.Schau at compaq.com)

Hard Core Coders: Riccardo Facchetti (riccardo at master.oasi.gpa.it)

Kern Sibbald (kern at sibbald dot com)

Part Time Coders: Jonathan H N Chin (jc254 at newton.cam.ac.uk)

Andre Hedrick (andre at linux-ide.org)

Brian Schau (Brian.Schau at compaq.com)

Carl Erhorn (Carl_Erhorn at hyperion.com)

Distribution Maintainers: Alpha: Kern Sibbald (kern at sibbald dot com) temp

Debian: Leon Breedt (ljb at debian.org)

FreeBSD/BSDi: Jeff Palmer (scorpio at drkshdw.org)

NetBSD: Neil Darlow (neil at darlow.co.uk)

HP-UX: Carl Erhorn (Carl_Erhorn at hyperion.com), Robert K Nelson (rnelson at airflowsciences.com)

OpenBSD: Devin Reade (gdr at gno.org)

RedHat: Kern Sibbald (kern at sibbald dot com)

Slackware: Devin Reade (gdr at gno.org)

Sparc Solaris: Carl Erhorn (Carl_Erhorn at hyperion.com)

SUSE: Riccardo Facchetti (riccardo at master.oasi.gpa.it)

Win32: Kern Sibbald (kern at sibbald dot com)

Project Discussions: APCUPSD Mailing List

Thanks to American Power Conversion (APC) who helped in giving technical information on their UPSes.

A special thanks to APC who gave me (Riccardo) a Smart UPS1400 INET when my old Back UPS v/s 650's battery died. Thank you guys, your help has been invaluable.

Thanks to all the users that send bug reports and suggestions:we need your help.

Thanks to every one I forgot here. If you feel Ihave forgot your name, please don't hesitate to tell me.

Miquel van Smoorenburg, The Doctor What, Pavel Korensky, and Russell Kroll <rkroll at exploits.org> for the CGI programs. Jonathan Benson <jbenson at technologist.com> for adapting the upsstatus.cgi program to work with apcupsd

The gd 1.2 Image Library used in our CGI programs is copyright 1994, 1995, Quest Protein Database Center, Cold Spring Harbor Labs. Permission granted to copy and distribute this work provided that this notice remains intact. Credit for the library must be given to the Quest Protein Database Center, Cold Spring Harbor Labs, in all derived works. This does not affect your ownership of the derived work itself, and the intent is to assure proper credit for Quest, not to interfere with your use of gd.

gd 1.2 was written by Thomas Boutell and is currently distributed by boutell.com, Inc.

Parts of the VNC project by ATT (cool code) were used as templates for our Win32 code, see: http://www.uk.research.att.com/vnc

A.2 Disclaimer: NO WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Appendix B Kernel Config

A typical USB section of a .config file might be:

```
#
# USB support
#
```

```
CONFIG_USB=m
CONFIG_USB_DEBUG=y

#
# Miscellaneous USB options
#
CONFIG_USB_DEVICEFS=y
# CONFIG_USB_BANDWIDTH is not set
# CONFIG_USB_DYNAMIC_MINORS is not set

#
# USB Host Controller Drivers
#
# CONFIG_USB_EHCI_HCD is not set
# CONFIG_USB_OHCI_HCD is not set
CONFIG_USB_UHCI_HCD=m

#
# USB Device Class drivers
#
# CONFIG_USB_BLUETOOTH_TTY is not set
# CONFIG_USB_ACM is not set
# CONFIG_USB_PRINTER is not set
CONFIG_USB_STORAGE=m
# CONFIG_USB_STORAGE_DEBUG is not set
# CONFIG_USB_STORAGE_DATAFAB is not set
# CONFIG_USB_STORAGE_FREECOM is not set
# CONFIG_USB_STORAGE_ISD200 is not set
# CONFIG_USB_STORAGE_DPCM is not set
# CONFIG_USB_STORAGE_HP8200e is not set
# CONFIG_USB_STORAGE_SDDR09 is not set
# CONFIG_USB_STORAGE_SDDR55 is not set
# CONFIG_USB_STORAGE_JUMPSHOT is not set

#
# USB Human Interface Devices (HID)
#
CONFIG_USB_HID=m
CONFIG_USB_HIDINPUT=y
# CONFIG_HID_FF is not set
CONFIG_USB_HIDDEV=y

#
# USB HID Boot Protocol drivers
#
# CONFIG_USB_KBD is not set
# CONFIG_USB_MOUSE is not set
# CONFIG_USB_AIPTEK is not set
# CONFIG_USB_WACOM is not set
# CONFIG_USB_KBTAB is not set
# CONFIG_USB_POWERMATE is not set
# CONFIG_USB_MTOUCH is not set
# CONFIG_USB_XPAD is not set
# CONFIG_USB_ATI_REMOTE is not set
```

```
#
# USB Imaging devices
#
# CONFIG_USB_MDC800 is not set
# CONFIG_USB_MICROTEK is not set
# CONFIG_USB_HPUSBSCSI is not set

#
# USB Multimedia devices
#
# CONFIG_USB_DABUSB is not set

#
# Video4Linux support is needed for USB Multimedia device support
#

#
# USB Network adaptors
#
# CONFIG_USB_CATC is not set
# CONFIG_USB_KAWETH is not set
# CONFIG_USB_PEGASUS is not set
# CONFIG_USB_RTL8150 is not set
# CONFIG_USB_USBNET is not set

#
# USB port drivers
#
# CONFIG_USB_USS720 is not set

#
# USB Serial Converter support
#
# CONFIG_USB_SERIAL is not set

#
# USB Miscellaneous drivers
#
# CONFIG_USB_EMI62 is not set
# CONFIG_USB_EMI26 is not set
# CONFIG_USB_TIGL is not set
# CONFIG_USB_AUERSWALD is not set
# CONFIG_USB_RIO500 is not set
# CONFIG_USB_LEGOTOWER is not set
# CONFIG_USB_LCD is not set
# CONFIG_USB_LED is not set
# CONFIG_USB_CYTHERM is not set
# CONFIG_USB_TEST is not set

#
# USB Gadget Support
#
# CONFIG_USB_GADGET is not set
```

Interpretation of /proc/usb info on 2.4 kernels:

```
/proc/bus/usb filesystem output
===============================
(version 2002.03.18)


The /proc filesystem for USB devices provides /proc/bus/usb/drivers
and /proc/bus/usb/devices, as well as /proc/bus/usb/BBB/DDD files.


**NOTE**: If /proc/bus/usb appears empty, and a host controller
          driver has been linked, then you need to mount the
          filesystem.  Issue the command (as root):

      mount -t usbfs none /proc/bus/usb

          An alternative and more permanent method would be to add

      none  /proc/bus/usb  usbfs  defaults  0  0

          to /etc/fstab.  This will mount usbfs at each reboot.
          You can then issue ‘cat /proc/bus/usb/devices‘ to extract
          USB device information, and user mode drivers can use usbfs
          to interact with USB devices.

          There are a number of mount options supported by usbfs.
          Consult the source code (linux/drivers/usb/inode.c) for
          information about those options.

**NOTE**: The filesystem has been renamed from "usbdevfs" to
          "usbfs", to reduce confusion with "devfs".  You may
          still see references to the older "usbdevfs" name.

For more information on mounting the usbfs file system, see the
"USB Device Filesystem" section of the USB Guide. The latest copy
of the USB Guide can be found at http://www.linux-usb.org/


THE /proc/bus/usb/BBB/DDD FILES:
--------------------------------
Each connected USB device has one file.  The BBB indicates the bus
number.  The DDD indicates the device address on that bus.  Both
of these numbers are assigned sequentially, and can be reused, so
you can’t rely on them for stable access to devices.  For example,
it’s relatively common for devices to re-enumerate while they are
still connected (perhaps someone jostled their power supply, hub,
or USB cable), so a device might be 002/027 when you first connect
it and 002/048 sometime later.

These files can be read as binary data.  The binary data consists
of first the device descriptor, then the descriptors for each
```

configuration of the device. That information is also shown in
text form by the /proc/bus/usb/devices file, described later.

These files may also be used to write user-level drivers for the USB
devices. You would open the /proc/bus/usb/BBB/DDD file read/write,
read its descriptors to make sure it's the device you expect, and then
bind to an interface (or perhaps several) using an ioctl call. You
would issue more ioctls to the device to communicate to it using
control, bulk, or other kinds of USB transfers. The IOCTLs are
listed in the linux/usbdevice_fs.h file, and at this writing the
source code (linux/drivers/usb/devio.c) is the primary reference
for how to access devices through those files.

Note that since by default these BBB/DDD files are writable only by
root, only root can write such user mode drivers. You can selectively
grant read/write permissions to other users by using "chmod". Also,
usbfs mount options such as "devmode=0666" may be helpful.

THE /proc/bus/usb/drivers FILE:

Each of the USB device drivers linked into your kernel (statically,
or dynamically using "modprobe") is listed in the "drivers" file.
Here's an example from one system:

        usbdevfs
        hub
  0- 15: usblp
        usbnet
        serial
        usb-storage
        pegasus

If you see this file, "usbdevfs" and "hub" will always be listed,
since those are part of the "usbcore" framework.

Drivers that use the USB major number (180) to provide character devices
will include a range of minor numbers, as shown above for the "usblp"
(actually "printer.o") module. USB device drivers can of course use any
major number, but it's easy to use the USB range since there's explicit
support for subdividing it in the USB device driver framework.

THE /proc/bus/usb/devices FILE:

In /proc/bus/usb/devices, each device's output has multiple
lines of ASCII output.
I made it ASCII instead of binary on purpose, so that someone
can obtain some useful data from it without the use of an
auxiliary program. However, with an auxiliary program, the numbers
in the first 4 columns of each "T:" line (topology info:
Lev, Prnt, Port, Cnt) can be used to build a USB topology diagram.

```
Each line is tagged with a one-character ID for that line:

T = Topology (etc.)
B = Bandwidth (applies only to USB host controllers, which are
    virtualized as root hubs)
D = Device descriptor info.
P = Product ID info. (from Device descriptor, but they won't fit
    together on one line)
S = String descriptors.
C = Configuration descriptor info. (* = active configuration)
I = Interface descriptor info.
E = Endpoint descriptor info.

======================================================================

/proc/bus/usb/devices output format:

Legend:
  d = decimal number (may have leading spaces or 0's)
  x = hexadecimal number (may have leading spaces or 0's)
  s = string


Topology info:

T:  Bus=dd Lev=dd Prnt=dd Port=dd Cnt=dd Dev#=ddd Spd=ddd MxCh=dd
|   |      |      |       |       |      |        |       |__MaxChildren
|   |      |      |       |       |      |        |__Device Speed in Mbps
|   |      |      |       |       |      |__DeviceNumber
|   |      |      |       |       |__Count of devices at this level
|   |      |      |       |__Connector/Port on Parent for this device
|   |      |      |__Parent DeviceNumber
|   |      |__Level in topology for this bus
|   |__Bus number
|__Topology info tag

    Speed may be:
        1.5     Mbit/s for low speed USB
        12      Mbit/s for full speed USB
        480     Mbit/s for high speed USB (added for USB 2.0)


Bandwidth info:
B:  Alloc=ddd/ddd us (xx%), #Int=ddd, #Iso=ddd
|   |                       |         |__Number of isochronous requests
|   |                       |__Number of interrupt requests
|   |__Total Bandwidth allocated to this bus
|__Bandwidth info tag

    Bandwidth allocation is an approximation of how much of one frame
    (millisecond) is in use.  It reflects only periodic transfers, which
    are the only transfers that reserve bandwidth.  Control and bulk
    transfers use all other bandwidth, including reserved bandwidth that
    is not used for transfers (such as for short packets).
```

```
    The percentage is how much of the "reserved" bandwidth is scheduled by
    those transfers.  For a low or full speed bus (loosely, "USB 1.1"),
    90% of the bus bandwidth is reserved.  For a high speed bus (loosely,
    "USB 2.0") 80% is reserved.


Device descriptor info \& Product ID info:

D:  Ver=x.xx Cls=xx(s) Sub=xx Prot=xx MxPS=dd #Cfgs=dd
P:  Vendor=xxxx ProdID=xxxx Rev=xx.xx

where
D:  Ver=x.xx Cls=xx(sssss) Sub=xx Prot=xx MxPS=dd #Cfgs=dd
|   |        |             |      |       |       |__NumberConfigurations
|   |        |             |      |       |__MaxPacketSize of Default Endpoint
|   |        |             |      |__DeviceProtocol
|   |        |             |__DeviceSubClass
|   |        |__DeviceClass
|   |__Device USB version
|__Device info tag #1

where
P:  Vendor=xxxx ProdID=xxxx Rev=xx.xx
|   |           |           |__Product revision number
|   |           |__Product ID code
|   |__Vendor ID code
|__Device info tag #2


String descriptor info:

S:  Manufacturer=ssss
|   |__Manufacturer of this device as read from the device.
|      For USB host controller drivers (virtual root hubs) this may
|      be omitted, or (for newer drivers) will identify the kernel
|      version and the driver which provides this hub emulation.
|__String info tag

S:  Product=ssss
|   |__Product description of this device as read from the device.
|      For older USB host controller drivers (virtual root hubs) this
|      indicates the driver; for newer ones, it's a product (and vendor)
|      description that often comes from the kernel's PCI ID database.
|__String info tag

S:  SerialNumber=ssss
|   |__Serial Number of this device as read from the device.
|      For USB host controller drivers (virtual root hubs) this is
|      some unique ID, normally a bus ID (address or slot name) that
|      can't be shared with any other device.
|__String info tag
```

```
Configuration descriptor info:

C:* #Ifs=dd Cfg#=dd Atr=xx MPwr=dddmA
| | |       |       |      |__MaxPower in mA
| | |       |       |__Attributes
| | |       |__ConfigurationNumber
| | |__NumberOfInterfaces
| |__ "*" indicates the active configuration (others are " ")
|__Config info tag
```

USB devices may have multiple configurations, each of which act rather differently. For example, a bus-powered configuration might be much less capable than one that is self-powered. Only one device configuration can be active at a time; most devices have only one configuration.

Each configuration consists of one or more interfaces. Each interface serves a distinct "function", which is typically bound to a different USB device driver. One common example is a USB speaker with an audio interface for playback, and a HID interface for use with software volume control.

```
Interface descriptor info (can be multiple per Config):

I:  If#=dd Alt=dd #EPs=dd Cls=xx(sssss) Sub=xx Prot=xx Driver=ssss
|   |      |      |       |              |      |       |__Driver name
|   |      |      |       |              |      |          or "(none)"
|   |      |      |       |              |      |__InterfaceProtocol
|   |      |      |       |              |__InterfaceSubClass
|   |      |      |       |__InterfaceClass
|   |      |      |__NumberOfEndpoints
|   |      |__AlternateSettingNumber
|   |__InterfaceNumber
|__Interface info tag
```

A given interface may have one or more "alternate" settings. For example, default settings may not use more than a small amount of periodic bandwidth. To use significant fractions of bus bandwidth, drivers must select a non-default altsetting.

Only one setting for an interface may be active at a time, and only one driver may bind to an interface at a time. Most devices have only one alternate setting per interface.

```
Endpoint descriptor info (can be multiple per Interface):

E:  Ad=xx(s) Atr=xx(ssss) MxPS=dddd Ivl=dddms
|   |        |            |         |__Interval (max) between transfers
|   |        |            |__EndpointMaxPacketSize
|   |        |__Attributes(EndpointType)
|   |__EndpointAddress(I=In,O=Out)
```

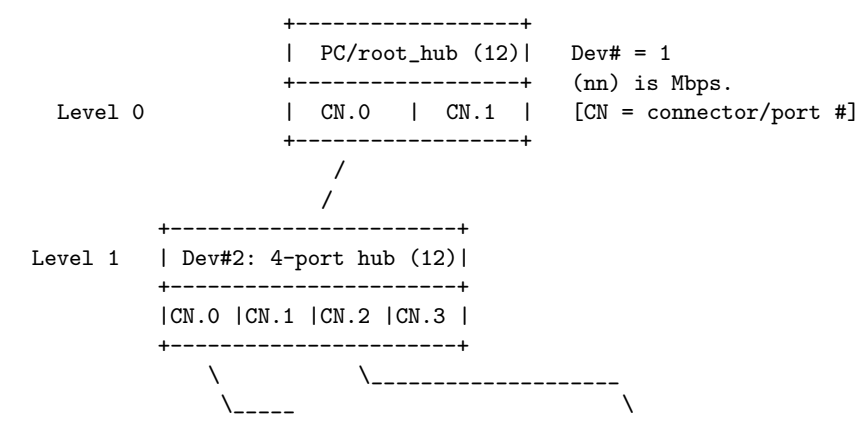
```
|__Endpoint info tag

    The interval is nonzero for all periodic (interrupt or isochronous)
    endpoints.  For high speed endpoints the transfer interval may be
    measured in microseconds rather than milliseconds.

    For high speed periodic endpoints, the "MaxPacketSize" reflects
    the per-microframe data transfer size.  For "high bandwidth"
    endpoints, that can reflect two or three packets (for up to
    3KBytes every 125 usec) per endpoint.

    With the Linux-USB stack, periodic bandwidth reservations use the
    transfer intervals and sizes provided by URBs, which can be less
    than those found in endpoint descriptor.


======================================================================


If a user or script is interested only in Topology info, for
example, use something like "grep ^T: /proc/bus/usb/devices"
for only the Topology lines.  A command like
"grep -i ^[tdp]: /proc/bus/usb/devices" can be used to list
only the lines that begin with the characters in square brackets,
where the valid characters are TDPCIE.  With a slightly more able
script, it can display any selected lines (for example, only T, D,
and P lines) and change their output format.  (The "procusb"
Perl script is the beginning of this idea.  It will list only
selected lines [selected from TBDPSCIE] or "All" lines from
/proc/bus/usb/devices.)

The Topology lines can be used to generate a graphic/pictorial
of the USB devices on a system's root hub.  (See more below
on how to do this.)

The Interface lines can be used to determine what driver is
being used for each device.

The Configuration lines could be used to list maximum power
(in milliamps) that a system's USB devices are using.
For example, "grep ^C: /proc/bus/usb/devices".


Here's an example, from a system which has a UHCI root hub,
an external hub connected to the root hub, and a mouse and
a serial converter connected to the external hub.

T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
B:  Alloc= 28/900 us ( 3%), #Int=  2, #Iso=  0
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 0.00
S:  Product=USB UHCI Root Hub
S:  SerialNumber=dce0
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
```

```
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=255ms
T:  Bus=00 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0451 ProdID=1446 Rev= 1.00
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   1 Ivl=255ms
T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=1.5 MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=04b4 ProdID=0001 Rev= 0.00
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=mouse
E:  Ad=81(I) Atr=03(Int.) MxPS=   3 Ivl= 10ms
T:  Bus=00 Lev=02 Prnt=02 Port=02 Cnt=02 Dev#=  4 Spd=12  MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0565 ProdID=0001 Rev= 1.08
S:  Manufacturer=Peracom Networks, Inc.
S:  Product=Peracom USB to Serial Converter
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 3 Cls=00(>ifc ) Sub=00 Prot=00 Driver=serial
E:  Ad=81(I) Atr=02(Bulk) MxPS=  64 Ivl= 16ms
E:  Ad=01(O) Atr=02(Bulk) MxPS=  16 Ivl= 16ms
E:  Ad=82(I) Atr=03(Int.) MxPS=   8 Ivl=  8ms
```

Selecting only the "T:" and "I:" lines from this (for example, by using "procusb ti"), we have:

```
T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
T:  Bus=00 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=1.5 MxCh= 0
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=mouse
T:  Bus=00 Lev=02 Prnt=02 Port=02 Cnt=02 Dev#=  4 Spd=12  MxCh= 0
I:  If#= 0 Alt= 0 #EPs= 3 Cls=00(>ifc ) Sub=00 Prot=00 Driver=serial
```

Physically this looks like (or could be converted to):



```
                    \                          \
              +--------------------+      +--------------------+
  Level 2     | Dev# 3: mouse (1.5)|      | Dev# 4: serial (12)|
              +--------------------+      +--------------------+
```

```
Or, in a more tree-like structure (ports [Connectors] without
connections could be omitted):

PC:  Dev# 1, root hub, 2 ports, 12 Mbps
|_ CN.0:  Dev# 2, hub, 4 ports, 12 Mbps
     |_ CN.0:  Dev #3, mouse, 1.5 Mbps
     |_ CN.1:
     |_ CN.2:  Dev #4, serial, 12 Mbps
     |_ CN.3:
|_ CN.1:


                        ### END ###
```

Interpretation of /proc/bus/usb info on 2.6 kernels:

```
/proc/bus/usb filesystem output
===============================
(version 2003.05.30)


The usbfs filesystem for USB devices is traditionally mounted at
/proc/bus/usb.  It provides the /proc/bus/usb/devices file, as well as
the /proc/bus/usb/BBB/DDD files.


**NOTE**: If /proc/bus/usb appears empty, and a host controller
          driver has been linked, then you need to mount the
          filesystem.  Issue the command (as root):

      mount -t usbfs none /proc/bus/usb

          An alternative and more permanent method would be to add

      none  /proc/bus/usb  usbfs  defaults  0  0

          to /etc/fstab.  This will mount usbfs at each reboot.
          You can then issue `cat /proc/bus/usb/devices` to extract
          USB device information, and user mode drivers can use usbfs
          to interact with USB devices.

          There are a number of mount options supported by usbfs.
          Consult the source code (linux/drivers/usb/core/inode.c) for
          information about those options.
```

```
**NOTE**: The filesystem has been renamed from "usbdevfs" to
          "usbfs", to reduce confusion with "devfs".  You may
          still see references to the older "usbdevfs" name.

For more information on mounting the usbfs file system, see the
"USB Device Filesystem" section of the USB Guide. The latest copy
of the USB Guide can be found at http://www.linux-usb.org/


THE /proc/bus/usb/BBB/DDD FILES:
--------------------------------
Each connected USB device has one file.  The BBB indicates the bus
number.  The DDD indicates the device address on that bus.  Both
of these numbers are assigned sequentially, and can be reused, so
you can't rely on them for stable access to devices.  For example,
it's relatively common for devices to re-enumerate while they are
still connected (perhaps someone jostled their power supply, hub,
or USB cable), so a device might be 002/027 when you first connect
it and 002/048 sometime later.

These files can be read as binary data.  The binary data consists
of first the device descriptor, then the descriptors for each
configuration of the device.  That information is also shown in
text form by the /proc/bus/usb/devices file, described later.

These files may also be used to write user-level drivers for the USB
devices.  You would open the /proc/bus/usb/BBB/DDD file read/write,
read its descriptors to make sure it's the device you expect, and then
bind to an interface (or perhaps several) using an ioctl call.  You
would issue more ioctls to the device to communicate to it using
control, bulk, or other kinds of USB transfers.  The IOCTLs are
listed in the linux/usbdevice_fs.h file, and at this writing the
source code (linux/drivers/usb/devio.c) is the primary reference
for how to access devices through those files.

Note that since by default these BBB/DDD files are writable only by
root, only root can write such user mode drivers.  You can selectively
grant read/write permissions to other users by using "chmod".  Also,
usbfs mount options such as "devmode=0666" may be helpful.


THE /proc/bus/usb/devices FILE:
-------------------------------
In /proc/bus/usb/devices, each device's output has multiple
lines of ASCII output.
I made it ASCII instead of binary on purpose, so that someone
can obtain some useful data from it without the use of an
auxiliary program.  However, with an auxiliary program, the numbers
in the first 4 columns of each "T:" line (topology info:
Lev, Prnt, Port, Cnt) can be used to build a USB topology diagram.

Each line is tagged with a one-character ID for that line:
```

```
T = Topology (etc.)
B = Bandwidth (applies only to USB host controllers, which are
    virtualized as root hubs)
D = Device descriptor info.
P = Product ID info. (from Device descriptor, but they won't fit
    together on one line)
S = String descriptors.
C = Configuration descriptor info. (* = active configuration)
I = Interface descriptor info.
E = Endpoint descriptor info.

======================================================================

/proc/bus/usb/devices output format:

Legend:
  d = decimal number (may have leading spaces or 0's)
  x = hexadecimal number (may have leading spaces or 0's)
  s = string


Topology info:

T:  Bus=dd Lev=dd Prnt=dd Port=dd Cnt=dd Dev#=ddd Spd=ddd MxCh=dd
|   |      |      |       |       |      |        |       |__MaxChildren
|   |      |      |       |       |      |        |__Device Speed in Mbps
|   |      |      |       |       |      |__DeviceNumber
|   |      |      |       |       |__Count of devices at this level
|   |      |      |       |__Connector/Port on Parent for this device
|   |      |      |__Parent DeviceNumber
|   |      |__Level in topology for this bus
|   |__Bus number
|__Topology info tag

    Speed may be:
        1.5     Mbit/s for low speed USB
        12      Mbit/s for full speed USB
        480     Mbit/s for high speed USB (added for USB 2.0)


Bandwidth info:
B:  Alloc=ddd/ddd us (xx%), #Int=ddd, #Iso=ddd
|   |                       |         |__Number of isochronous requests
|   |                       |__Number of interrupt requests
|   |__Total Bandwidth allocated to this bus
|__Bandwidth info tag

    Bandwidth allocation is an approximation of how much of one frame
    (millisecond) is in use.  It reflects only periodic transfers, which
    are the only transfers that reserve bandwidth.  Control and bulk
    transfers use all other bandwidth, including reserved bandwidth that
    is not used for transfers (such as for short packets).
```

```
    The percentage is how much of the "reserved" bandwidth is scheduled by
    those transfers.  For a low or full speed bus (loosely, "USB 1.1"),
    90% of the bus bandwidth is reserved.  For a high speed bus (loosely,
    "USB 2.0") 80% is reserved.


Device descriptor info \& Product ID info:

D:  Ver=x.xx Cls=xx(s) Sub=xx Prot=xx MxPS=dd #Cfgs=dd
P:  Vendor=xxxx ProdID=xxxx Rev=xx.xx

where
D:  Ver=x.xx Cls=xx(sssss) Sub=xx Prot=xx MxPS=dd #Cfgs=dd
|   |        |             |      |       |       |__NumberConfigurations
|   |        |             |      |       |__MaxPacketSize of Default Endpoint
|   |        |             |      |__DeviceProtocol
|   |        |             |__DeviceSubClass
|   |        |__DeviceClass
|   |__Device USB version
|__Device info tag #1

where
P:  Vendor=xxxx ProdID=xxxx Rev=xx.xx
|   |           |           |__Product revision number
|   |           |__Product ID code
|   |__Vendor ID code
|__Device info tag #2


String descriptor info:

S:  Manufacturer=ssss
|   |__Manufacturer of this device as read from the device.
|      For USB host controller drivers (virtual root hubs) this may
|      be omitted, or (for newer drivers) will identify the kernel
|      version and the driver which provides this hub emulation.
|__String info tag

S:  Product=ssss
|   |__Product description of this device as read from the device.
|      For older USB host controller drivers (virtual root hubs) this
|      indicates the driver; for newer ones, it's a product (and vendor)
|      description that often comes from the kernel's PCI ID database.
|__String info tag

S:  SerialNumber=ssss
|   |__Serial Number of this device as read from the device.
|      For USB host controller drivers (virtual root hubs) this is
|      some unique ID, normally a bus ID (address or slot name) that
|      can't be shared with any other device.
|__String info tag
```

```
Configuration descriptor info:

C:* #Ifs=dd Cfg#=dd Atr=xx MPwr=dddmA
| | |       |       |      |__MaxPower in mA
| | |       |       |__Attributes
| | |       |__ConfiguratioNumber
| | |__NumberOfInterfaces
| |__ "*" indicates the active configuration (others are " ")
|__Config info tag

    USB devices may have multiple configurations, each of which act
    rather differently.  For example, a bus-powered configuration
    might be much less capable than one that is self-powered.  Only
    one device configuration can be active at a time; most devices
    have only one configuration.

    Each configuration consists of one or more interfaces.  Each
    interface serves a distinct "function", which is typically bound
    to a different USB device driver.  One common example is a USB
    speaker with an audio interface for playback, and a HID interface
    for use with software volume control.


Interface descriptor info (can be multiple per Config):

I:  If#=dd Alt=dd #EPs=dd Cls=xx(sssss) Sub=xx Prot=xx Driver=ssss
|   |      |      |       |              |      |       |__Driver name
|   |      |      |       |              |      |          or "(none)"
|   |      |      |       |              |      |__InterfaceProtocol
|   |      |      |       |              |__InterfaceSubClass
|   |      |      |       |__InterfaceClass
|   |      |      |__NumberOfEndpoints
|   |      |__AlternateSettingNumber
|   |__InterfaceNumber
|__Interface info tag

    A given interface may have one or more "alternate" settings.
    For example, default settings may not use more than a small
    amount of periodic bandwidth.  To use significant fractions
    of bus bandwidth, drivers must select a non-default altsetting.

    Only one setting for an interface may be active at a time, and
    only one driver may bind to an interface at a time.  Most devices
    have only one alternate setting per interface.


Endpoint descriptor info (can be multiple per Interface):

E:  Ad=xx(s) Atr=xx(ssss) MxPS=dddd Ivl=dddss
|   |        |            |         |__Interval (max) between transfers
|   |        |            |__EndpointMaxPacketSize
|   |        |__Attributes(EndpointType)
|   |__EndpointAddress(I=In,O=Out)
|__Endpoint info tag
```

```
    The interval is nonzero for all periodic (interrupt or isochronous)
    endpoints.  For high speed endpoints the transfer interval may be
    measured in microseconds rather than milliseconds.

    For high speed periodic endpoints, the "MaxPacketSize" reflects
    the per-microframe data transfer size.  For "high bandwidth"
    endpoints, that can reflect two or three packets (for up to
    3KBytes every 125 usec) per endpoint.

    With the Linux-USB stack, periodic bandwidth reservations use the
    transfer intervals and sizes provided by URBs, which can be less
    than those found in endpoint descriptor.
```

```
======================================================================
```

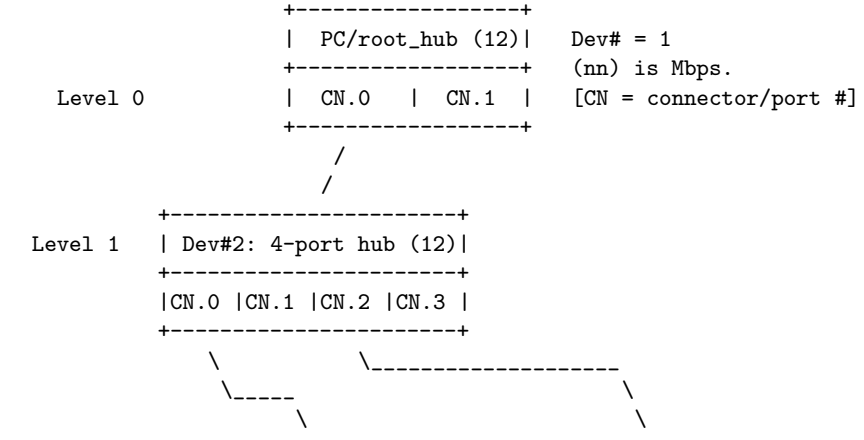
```
If a user or script is interested only in Topology info, for
example, use something like "grep ^T: /proc/bus/usb/devices"
for only the Topology lines.  A command like
"grep -i ^[tdp]: /proc/bus/usb/devices" can be used to list
only the lines that begin with the characters in square brackets,
where the valid characters are TDPCIE.  With a slightly more able
script, it can display any selected lines (for example, only T, D,
and P lines) and change their output format.  (The "procusb"
Perl script is the beginning of this idea.  It will list only
selected lines [selected from TBDPSCIE] or "All" lines from
/proc/bus/usb/devices.)

The Topology lines can be used to generate a graphic/pictorial
of the USB devices on a system's root hub.  (See more below
on how to do this.)

The Interface lines can be used to determine what driver is
being used for each device.

The Configuration lines could be used to list maximum power
(in milliamps) that a system's USB devices are using.
For example, "grep ^C: /proc/bus/usb/devices".


Here's an example, from a system which has a UHCI root hub,
an external hub connected to the root hub, and a mouse and
a serial converter connected to the external hub.

T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
B:  Alloc= 28/900 us ( 3%), #Int=  2, #Iso=  0
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 0.00
S:  Product=USB UHCI Root Hub
S:  SerialNumber=dce0
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
```

```
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=255ms
T:  Bus=00 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0451 ProdID=1446 Rev= 1.00
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   1 Ivl=255ms
T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=1.5 MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=04b4 ProdID=0001 Rev= 0.00
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=mouse
E:  Ad=81(I) Atr=03(Int.) MxPS=   3 Ivl= 10ms
T:  Bus=00 Lev=02 Prnt=02 Port=02 Cnt=02 Dev#=  4 Spd=12  MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0565 ProdID=0001 Rev= 1.08
S:  Manufacturer=Peracom Networks, Inc.
S:  Product=Peracom USB to Serial Converter
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 3 Cls=00(>ifc ) Sub=00 Prot=00 Driver=serial
E:  Ad=81(I) Atr=02(Bulk) MxPS=  64 Ivl= 16ms
E:  Ad=01(O) Atr=02(Bulk) MxPS=  16 Ivl= 16ms
E:  Ad=82(I) Atr=03(Int.) MxPS=   8 Ivl=  8ms
```

Selecting only the "T:" and "I:" lines from this (for example, by using "procusb ti"), we have:

```
T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
T:  Bus=00 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=1.5 MxCh= 0
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=mouse
T:  Bus=00 Lev=02 Prnt=02 Port=02 Cnt=02 Dev#=  4 Spd=12  MxCh= 0
I:  If#= 0 Alt= 0 #EPs= 3 Cls=00(>ifc ) Sub=00 Prot=00 Driver=serial
```

Physically this looks like (or could be converted to):



```
              +--------------------+      +--------------------+
  Level 2     | Dev# 3: mouse (1.5)|      | Dev# 4: serial (12)|
              +--------------------+      +--------------------+
```

Or, in a more tree-like structure (ports [Connectors] without connections could be omitted):

```
PC:  Dev# 1, root hub, 2 ports, 12 Mbps
|_ CN.0:  Dev# 2, hub, 4 ports, 12 Mbps
     |_ CN.0:  Dev #3, mouse, 1.5 Mbps
     |_ CN.1:
     |_ CN.2:  Dev #4, serial, 12 Mbps
     |_ CN.3:
|_ CN.1:
```

END

Index

Index

/about , 125
/events , 125
/help , 125
/install , 125
/kill , 125
/remove , 125
/run , 125
/service , 125
/servicehelper , 124
/status , 125
=appendix¿Appendix A Credits , 191
—disable-install-distdir , 33
—enable-apcsmart , 31
—enable-cgi , 31
—enable-dumb , 31
—enable-master-slave , 31
—enable-net , 31
—enable-nis , 31
—enable-powerflute , 31
—enable-pthreads , 32
—enable-snmp , 31
—enable-test , 31
—enable-usb , 31
—prefix , 30
—sbindir , 30
—with-cgi-bin , 31
—with-css-dir , 31
—with-libwrap , 32
—with-lock-dir , 32
—with-log-dir , 32
—with-net-port , 33
—with-nis-port , 32
—with-nisip , 33
—with-nologin , 32
—with-pid-dir , 32
—with-pwrfail-dir , 32
—with-serial-dev , 32
—with-upscable , 33
—with-upstype , 33
2.0 User's Manual: , 192

A: , 88–93
Q: , 88–93

a name , 88
A.1 Contributors , 192
A.2 Disclaimer: NO WARRANTY , 194
Acknowledgements , 189
Advanced topics , 94
After Installation , 43
ALARMDEL , 169
Alert messages , 183
Alpha Port: , 192
Alternate Ways To Run The Network Information Server , 108
always , 157
AMBTEMP , 170
Analyses of APC Cables , 134
ANNOY <time in seconds> , 156
ANNOYDELAY <time in seconds> , 156
annoyme , 95
Answers , 189
APC , 167
APC smart protocol , 176
Apcaccess , 66
Apcaccess Test , 55
Apccontrol Command Line Options , 95

apcmain , 53
APCMODEL , 170
apcnis , 53
apcser , 53
apcsmart , 12
Apctest , 62
Apcupsd
    Building and Installing , 26, 107
    Testing , 53
Apcupsd Bugs , 94
Apcupsd Known USB Issues , 15
Apcupsd Manual , 6
Apcupsd Network Monitoring (CGI) Programs , 70
Apcupsd Notification and Events , 69
Apcupsd Status Logging , 165
Apcupsd Support and Knowledge Base: , 192
Apcupsd System Logging , 111
Apcupsd — RPM Packaging FAQ , 189
Appendix B Kernel Config , 195
Areas
    Problem , 120
Arranging for Reboot on Power-Up , 44

Back-UPS Office 500 signals , 133
Bad APC magic from master: MMM , 103
Bad user magic from master: MMM , 103
BADBATTS , 170
Basic User's Guide , 9
BATTDATE , 169
BATTDATE <string> , 165
Batteries
    What Various People Have to Say about , 81
    Where Carl Suggests You Get , 87
BATTERYLEVEL <percent of battery> , 157
BATTV , 57, 169
BCHARGE , 168
BEEPSTATE <alarm beep state> , 164
Beginners
    Quick Start for , 9
Bits
    Status , 182
Bugs
    Apcupsd , 94
Building and Installing apcupsd , 26, 107
Building the Win32 Version from the Source , 125

CABLE , 167
Cables
    Analyses of APC , 134
    Internal Apcupsd Actions for Simple , 141
    Supported Operating Systems UPSes and , 11
    Voltage Signalling Features Supported by Apcupsd for Various , 132
Cables , 127
Caldera: , 192
Cannot bind local address, probably already in use , 102
Cannot open stream socket , 102
Cannot read magic from slave SSS , 101
Cannot resolve master name MMM , 102
Cannot resolve slave name XXX , 101
Cannot write to slave SSS , 101
CGI and HTML fixer: , 192
changeme , 96
Checking Out Your USB Subsystem , 17

Checking Your Configuration File , 43
Command Line Options Specific to the Windows Version , 124
commfailure , 96
commok , 96
Communications
Once you have established serial , 150
Troubleshooting Serial Line , 150
Communications Test , 58
Compilers and Options , 34
Config
Appendix B Kernel , 195
Configuration
Simple Master , 49
Simple Slave , 49
Simple USB , 47
Variation on the Master/Slave , 50
Configuration , 104
Configuration Directive Reference , 153
Configuration Directives , 99
Configuration Directives for Sharing a UPS , 159
Configuration Directives Used by the Network Information Server , 155
Configuration Directives used during Power Failures , 156
Configuration Directives used to Control System Logging , 159
Configuration Directives Used to Set the UPS EPROM , 163
Configuration Examples , 47
Configurations
Master/Slave , 99
Network Problems with Master/Slave or Server/Slave , 101
Configure Options , 30
Configuring Your EEPROM , 77
Connect to slave SSS failed , 102
connected , 95
Connecting a APC USB UPS to either a PC USB or Serial Port , 126
Connecting a Serial-Line UPS to a USB Port , 126
Connecting an SNMP UPS , 106
Connection
Establishing Serial Port , 145
Considerations
Windows , 176
Contributors
A.1 , 192
Controlling Multiple UPSes on one Machine , 104
Conventions
RS232 Wiring and Signal , 143
Correspondence
Ioctl to RS232 , 144
Credits
=appendix¿Appendix A , 191
CUMONBATT , 169
Current Code Maintainer and Project Manager: , 192
Customizing Event Handling , 94

DATA Logging , 111, 153
DATATIME<time> , 159
DATE , 167
Description , 176
Details
Implementation , 112
Determining Which Voltage-Signaling Cable You Have , 150
Developers Notes , 113
DEVICE <name of device> , 154
Diagram for cable hackers , 177
Differences
RS-232 , 176

Dip switch info , 182
DIPSW , 169
Directives
    Configuration , 99
    General Configuration , 153
Directory
    Installation , 118
disable , 157
Disclaimer , 121
Discontents
    Shutown Sequence and its , 171
Distribution Maintainers: , 193
DLOWBATT , 168
doreboot , 96
doshutdown , 96
Driver
    Sample NIS Slave Configuration Using the Net , 50
DSHUTD , 168
Dumb
    Simple Configuration for a Simple Signaling or , 48
dumb , 12
DWAKE , 168

EEPROM
    Configuring Your , 77
    Programming the UPS , 188
EEPROM Values , 187
Email Notification of Events , 122
emergency , 96
End
    Pin Assignment for the Serial Port RS-232C 25-pin and 9-pin Female , 143
END APC , 170
endselftest , 98
EPROM
    Configuration Directives Used to Set the UPS , 163
Error
    Socket accept , 102
Establishing Serial Port Connection , 145
event , 95
Events
    Apcupsd Notification and , 69
    Email Notification of , 122
EVENTS Logging , 111
EVENTSFILE <filename> , 155
Example
    Status Report , 166
Examples
    Configuration , 47
EXTBATTS , 170

FACILITY<log-facility> , 159
Failed
    Connect to slave SSS , 102
failing , 96
Failures
    Configuration Directives used during Power , 156
FAQ
    Apcupsd — RPM Packaging , 189
Features
    New , 6
Fields
    Status Report , 167
File
    Checking Your Configuration , 43
FIRMWARE , 170
Format
    Status report , 165
Former Code Maintainer and Project Manager: , 192
Frequently-Asked Questions , 88
Full Power Down Test , 61
Functions
    Utility , 121

General Configuration Directives , 153
Got slave shutdown from SSS , 101

Guide
Basic User's , 9

Hackers
Diagram for cable , 177
Handling
Customizing Event , 94
Hard Core Coders: , 192
Have
Determining Which Voltage-Signaling Cable You , 150
Hid-ups and USB Specific Information , 69
HITRANS , 169
HITRANSFER <upper limit of ups batt. transfer> , 164
HOSTNAME , 167
How do I build Apcupsd for platform xxx? , 189
How do I control whether usb support gets built? , 190
How To Use This Manual , 9
HP-UX Port , 192
HUMIDITY , 170

I'm getting errors about not having permission when , 191
Implementation Details , 112
Index , 211
INETD
Running apcnisd from , 109
Info
Dip switch , 182
Information
hid-ups and USB Specific , 69
Logging the STATUS , 170
SNMP Specific , 107
Installation
After , 43
Planning Your , 9
Post , 119
Troubleshooting Your , 64
Verifying a Source , 28
Installation , 114
Installation Directory , 118
Installation from Binary Packages , 26
Installation from Source , 26
Installation: Serial-Line UPSes , 125
Installation: Windows , 113
Internal Apcupsd Actions for Simple Cables , 141
Interpretation of the New Firmware Revision , 186
Interpretation of the Old Firmware Revision , 185
Ioctl to RS232 Correspondence , 144
Issues
Apcupsd Known USB , 15
ITEMP , 169

KILLDELAY <time in seconds> , 158
Killpower under Windows , 123
Known Problems , 107
Known Problems with USB UPSes , 64

LASTXFER , 169
LINEFREQ , 169
LINEV , 57, 168
loadlimit , 96
LOADPCT , 168
LOCKFILE <path to lockfile> , 154
Logging
apcupsd Status , 165
apcupsd System , 111
Configuration Directives used to Control System , 159
DATA , 111, 153
EVENTS , 111
STATUS , 111
Logging Test , 54

Logging the STATUS Information , 170
Logging Types , 111
LOTRANS , 168
LOTRANSFER <lower limit of ups batt. transfer> , 164

Machine
  Controlling Multiple UPSes on one , 104
mainsback , 96
Maintaining Your UPS , 80
Making sure apcupsd Is Running , 45
MANDATE , 169
Manual
  Apcupsd , 6
  How To Use This , 9
MASTER <name of the master> for slave machines. , 161
Master/Slave Configurations , 99
Master/Slave Problems , 99
Master/Slave Shutdown , 174
masterconnect , 98
mastertimeout , 98
MASTERUPD , 168
MAXLINEV , 168
MAXTIME , 168
MBATTCHG , 168
Messages
  Alert , 183
MINLINEV , 168
MINTIMEL , 168
minutes , 157
MINUTES <battery runtime in minutes> , 157
MMM
  Bad APC magic from master: , 103
  Bad user magic from master: , 103
  Cannot resolve master name , 102
  Unauthorized attempt from master , 103
MODEL , 167
Models
  Smart Signalling Cable for BackUPS CS , 128
Monitoring and Tuning your UPS , 66

net , 12
NetBSD Port: , 192
NETPORT <IP port number> , 161
NETSERVER [on , off] 155
NETTIME <time in seconds> , 161
Network Problems with Master/Slave or Server/Slave Configurations , 101
New Features , 6
NISIP <IP-address> , 155
NISPORT <port> , 155
NOLOGON <specifies when apcupsd should prevent user logins> , 156
NOMBATTV , 170
NOMOUTV , 170
Notes
  Developers , 113
  Release , 6
NUMXFERS , 169

offbattery , 97
onbattery , 97
Once you have established serial communications , 150
OpenBSD Port: , 192
Operating System Specifics , 34
Options
  apccontrol Command Line , 95
  Compilers and , 34
  Configure , 30

Other APC Cables that apcupsd Supports , 132
other defines are used? , 190
OUTPUTV , 168
Overview of Serial-Interface UPSes , 125

Packages
  Installation from Binary , 26
Part Time Coders: , 193
percent , 157
Pin Assignment for the Serial Port (RS-232C), 25-pin and 9-pin, Female End , 143
Planning Your Installation , 9
Port
  Connecting a APC USB UPS to either a PC USB or Serial , 126
  Connecting a Serial-Line UPS to a USB , 126
Post Installation , 119
Power Down During Shutdown , 124
powered , 95
powerout , 97
Problem Areas , 120
Problems
  Known , 107
  Master/Slave , 99
  Shutdown , 174
Process-Status Test , 53
Programming the UPS EEPROM , 188
Programs
  apcupsd Network Monitoring CGI , 70
  Setting up and Testing the CGI , 70
Project Discussions: , 193
Project Starter and Former Code Maintainer: , 192
Protocol
  APC smart , 176
  Smart , 177

Questions
  Frequently-Asked , 88
Quick Start for Beginners , 9

Read failure from socket , 103
Recalibrating the UPS Runtime , 151
Recommended Options for most Systems , 33
Reference
  Configuration Directive , 153
  Technical , 153
REG1 , 169
REG2 , 169
REG3 , 169
Register 1 , 184
Register 2 , 184
Register 3 , 185
RELEASE , 167
Release Notes , 6
remotedown , 97
restartme , 97
RETPCT , 169
RETURNCHARGE <min. batt. charge level> , 164
Revision
  Interpretation of the New Firmware , 186
  Interpretation of the Old Firmware , 185
RPM Packager: , 192
RS-232 differences , 176
RS232 Wiring and Signal Conventions , 143
runlimit , 97
Running
  Making sure apcupsd Is , 45
Running apcnisd from INETD , 109
Running apcnisd Standalome , 110

Running the server as a child of apcupsd , 108
Runtime
Recalibrating the UPS , 151

Sample NIS Slave Configuration Using the Net Driver , 50
SELFTEST , 169
SENSE , 168
SENSITIVITY <sets sensitivity level> , 163
Sequence
Shutdown , 62, 171
Serial Communications: , 192
SERIALNO , 169
Server
Alternate Ways To Run The Network Information , 108
Configuration Directives Used by the Network Information , 155
Setting up and Testing the CGI Programs , 70
Shutdown
Master/Slave , 174
Power Down During , 124
Shutdown Problems , 174
Shutdown Sequence , 62, 171
Shutown Sequence and its Discontents , 171
Signalling
Voltage , 133
Signals
Back-UPS Office 500 , 133
Simple Configuration for a Simple Signaling or Dumb , 48
Simple Configuration for a SmartUPS , 48
Simple Master Configuration , 49
Simple Slave Configuration , 49
Simple USB Configuration , 47
Simulated Power Fail Test , 59
SLAVE <name of slave(s)> used only in MASTER configuration files. , 162
SLEEP <set sleep delay> , 163
Smart Protocol , 177
Smart Signalling Cable for Back-UPS CS Models , 128
Smart-Custom Cable for SmartUPSes , 127
SmartUPS
Simple Configuration for a , 48
SmartUPSes
Smart-Custom Cable for , 127
snmp , 12
SNMP Specific Information , 107
Socket
Cannot open stream , 102
Read failure from , 103
Socket accept error , 102
SOLARIS Port: , 192
Source
Building the Win32 Version from the , 125
Installation from , 26
Specifics
Operating System , 34
SSS
Cannot read magic from slave , 101
Cannot write to slave , 101
Got slave shutdown from , 101
Standalome
Running apcnisd , 110
startselftest , 98
STARTTIME , 168
Startup , 175
STATFILE <file> , 159
STATFLAG , 169
STATTIME<time> , 159
STATUS , 57, 168
Status bits , 182
STATUS Logging , 111

Status Logging On Serial-Line UPSes , 152
Status Report Example , 166
Status Report Fields , 167
Status report format , 165
STESTI , 169
Subsystem
Checking Out Your USB , 17
Super USB bug fixer: , 192
Support for SNMP UPSes , 106
Supported Operating Systems, UPSes and Cables , 11
Supports
Other APC Cables that apcupsd , 132
System Shutdown Test , 61
Systems
Recommended Options for most , 33

TCP Wrappers , 77
Technical Reference , 153
Test
apcaccess , 55
Communications , 58
Full Power Down , 61
Logging , 54
Process-Status , 53
Simulated Power Fail , 59
System Shutdown , 61
Testing , 118
Testing Apcupsd , 53
Testing Serial-Line UPSes , 144
TIMELEFT , 168
timeout , 98, 157
TIMEOUT <time in seconds> , 158
TONBATT , 169
Topics
Advanced , 94
Troubleshooting Serial Line communications , 150
Troubleshooting Your Installation , 64
Types
Logging , 111

Unauthorized attempt from master MMM , 103
Upgrading , 119
UPS
Configuration Directives for Sharing a , 159
Connecting an SNMP , 106
Maintaining Your , 80
Monitoring and Tuning your , 66
ups-name , 95
UPSCABLE <type of cable you are using>> , 154
UPSCLASS <class of operation> , 160
UPSes
Installation: Serial-Line , 125
Known Problems with USB , 64
Overview of Serial-Interface , 125
Status Logging On Serial-Line , 152
Support for SNMP , 106
Testing Serial-Line , 144
Voltage-Signalling Cable for dumb , 129
Win32 Implementation Restrictions for Simple , 141
UPSMODE , 167
UPSMODE [disable , share 160
UPSNAME , 168
UPSNAME <string> , 165
UPSses
Using apctest on Serial-Line , 148
UPSTYPE <type of APC UPS you have> , 154
usb , 12
Use

Cannot bind local address probably already in , 102
USERMAGIC < user defined magic> used only in SLAVE configuration files. , 162
Using apctest on Serial-Line UPSses , 148
Utility Functions , 121

Values
EEPROM , 187
Variation on the Master/Slave Configuration , 50
Verifying a Source Installation , 28
Version
Command Line Options Specific to the Windows , 124
Voltage Signalling , 133
Voltage Signalling Features Supported by Apcupsd for Various Cables , 132
Voltage-Signalling Cable for dumb UPSes , 129

WAKEUP <set wakeup delay> , 163
WARRANTY
A.2 Disclaimer: NO , 194
WEB Interfaces: , 192
What Various People Have to Say about Batteries , 81
Where Carl Suggests You Get Batteries , 87
Win32 Implementation Restrictions for Simple UPSes , 141
Win32 Port: , 192
Windows
Installation: , 113
Killpower under , 123
Windows Considerations , 176
Windows Version of apcupsd , 113
Wrappers
TCP , 77

XOFFBATT , 169
XONBATT , 169
XXX
Cannot resolve slave name , 101