Netatalk 2.0 Manual

25th June 2004

Contents

Chapter 1

Introduction to Netatalk

Netatalk is a collection of server programs and utilities for handling various protocols employed by Apple Macintosh computers on Unix compatible systems.

This allows Unix hosts to act as file, print, and time servers for Apple Macintosh (classic MacOS as well as MacOS X) computers.

1.1 Background

Not yet ...

Chapter 2

Installation

WARNING

If you have previously used an older version of Netatalk, please read the chapter about upgrading first !!!

2.1 How to obtain Netatalk

Please have a look at the netatalk page on sourceforge for the most recent informations on this issue.
<http://sourceforge.net/projects/netatalk/>

2.1.1 Binary packages

Binary packages of Netatalk are included in some Linux and UNIX distributions.

2.1.2 Source packages

2.1.2.1 Tarballs

Prepacked tarballs in .tar.gz and tar.bz2 format are available on the netatalk page on sourceforge

2.1.2.2 Anonymous CVS

Downloading of the CVS source can be done quickly and easily.

1. Make sure you have cvs installed. **which cvs** should produce a path to cvs.

```
$> which cvs
/usr/bin/cvs
```

2. If you don't have one make a source directory. **cd** to this directory.

```
$> mkdir /path/to/new/source/dir
$> cd /path/to/new/source/dir
```

3. Authenticate yourself with cvs. Just hit enter for the password for the anonymous user.

```
$> cvs -d:pserver:anonymous@cvs.netatalk.sourceforge.net:/cvsroot/netatalk login
Logging in to :pserver:anonymous@cvs.netatalk.sourceforge.net:2401/cvsroot/netatalk
CVS password: [Enter]
```

4. Now get the source:

```
$> cvs -z3 -d:pserver:anonymous@cvs.netatalk.sourceforge.net:/cvsroot/netatalk co netatalk
cvs server: Updating netatalk
U netatalk/.cvsignore
U netatalk/CONTRIBUTORS
U netatalk/COPYING
...
```

   This will create a netatalk directory and download a complete and fresh copy of the netatalk source from the CVS repository.

5. Now **cd** to the netatalk directory and run **./autogen.sh**. This will create the `configure` script required in the next step.

```
$> ./autogen.sh
```

2.2 Compiling Netatalk

2.2.1 Prequisites

2.2.1.1 System requirements

You system needs to meet the following requirements:

- A C compiler, Netatalk compiles fine with gcc > 2.7.95

To be able to compile with AFP3 support, your system has to support large files (>2GB).

NOTE

On linux systems glibc > 2.2 is required.

2.2.1.2 Required third party software

Netatalk makes use of sleepycats' Berkeley DB. At the time of writing, the following versions are supported:

- 4.1.25
- 4.2.52 (recommended)

In case Berkeley DB is not installed on your system, please download it from:

<http://www.sleepycat.com/download/patchlogs.shtml>

and follow the installation instructions.

2.2.1.3 Optional third party software

Netatalk can use the following third party software to enhance it functionality.

- OpenSSL (recommended)

  OpenSSL is required for encrypted passwords. Without it, password will be sent over the network in cleartext.

- TCP wrappers

  Wietse Venema's network logger, also known as TCPD or LOG_TCP.

  Security options are: access control per host, domain and/or service; detection of host name spoofing or host address spoofing; booby traps to implement an early-warning system.

  TCP Wrappers can be downloaded from: <ftp://ftp.porcupine.org/pub/security>/

- PAM

  PAM provides a flexible mechanism for authenticating users. PAM was invented by SUN Microsystems. Linux-PAM is a suite of shared libraries that enable the local system administrator to choose how applications authenticate users.

  You can get the Linux PAM documentation and sources from <http://www.kernel.org/pub/linux/libs/pam/>

- OpenSLP

  SLP (Service Location Protocol) is an IETF standards track protocol that provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks.

  Mac OS X uses it to locate AFP servers, even though newer version prefer Rendevouz.

  You can download OpenSLP from: <http://www.openslp.org/>

- iconv

  iconv provides conversion routines for many character encodings. Netatalk uses it to provide charsets it does not have built in coversions for, like ISO-8859-1. On glibc systems, Netatalk can use the glibc provided iconv implementation. Otherwise you can use the GNU libiconv implementation.

  You can download GNU libiconv from: <http://www.gnu.org/software/libiconv/>

2.2.2 Compiling Netatalk

2.2.2.1 Configuring the build

To build the binaries, first run the program **./configure** in the source directory. This should automatically configure Netatalk for your operating system. If you have unusual needs, then you may wish to run

```
$> ./configure --help
```

to see what special options you can enable.

The most used configure options are:

- –enable-[redhat/suse/cobalt/netbsd/fhs]

  This option helps netatalk to determine where to install the start scripts.

- `-with-bdb=`*`/path/to/bdb/installation/`*

  In case you installed Berkeley DB in a non-standard location, you will *have* to give the install location to netatalk, using this switch.

Now run configure with any options you need

```
$> ./configure [arguments] [--with-bdb=/bdb/install/path]
```

If this step fails please visit the troubleshooting guide

NOTE

With recent RedHat releases, Berkeley DB links to libpthread. Netatalk does not link to libpthread, so detection of Berkely DB fails when running configure. It's recommended to (re)compile Berkeley DB with `-with-mutex="x86/gcc-assembly"` (on x86 platforms) to disable the use of libpthread. Alternatively you could use

```
$> LIBS="-lpthread" ./configure [arguments]
```

to trick Netatalk into linking to libpthread. However, this is not recommended, as there have been some trouble reports indicating that linking to libpthread badly damages performance.

Next, running

```
$> make
```

should produce the Netatalk binaries.

When the process finished you can use

```
$> make install
```

to install the binaries and documentation.

2.2.3 Compiling a new Berkeley DB for Netatalk

Netatalk 2.0 requires Berkeley DB version 4.1.25 or newer. Even if you already have a supported version of Berkeley DB installed on your system, there are several reasons, why you might still want to consider building a new version for Netatalk.

Many linux distributions provide a precompiled Berkeley DB version. This is usually nice, but also has one major drawback: If you update your system to a newer release, the installed version of Berkeley DB may change. This can lead to a number of problems, starting with strange behaviour of Netatalk, unreadable CNID Databases. Most likely Netatalk(afpd) won't start anymore, so you'll have to recompile Netatalk.

For instructions compiling Berkeley DB, you should generally refer to the documentation provided by Sleepycat. The following information is meant to help you avoid problems, users experienced in the past.

In case you are building on a recent RedHat release, please use `-with-mutex="x86/gcc-assembly"` on x86 platforms to prevent Berkeley DB from linking against libpthread.

2.2.3.1 Using a statically linked Berkeley DB

To link Netatalk statically to Berkeley DB, you have to disable shared libraries when building Berkeley DB. If shared libraries exist, Netatalk will always link to them, even if a static version exists in the same location.

```
root# cd build_unix
root# ../dist/configure --prefix=/install/path --disable-shared
root# make
root# make install
```

You should now continue with building Netatalk.

2.2.3.2 Using a dynamically linked Berkeley DB

Building a shared version of Berkeley DB is rather straightforward. However, especially under Linux, some care needs to be taken. Underlying system libraries, i.e. libnss_db, might be using Berkeley DB as well. As these libraries have likely been build with another, i.e. older, version of Berkeley DB, linking afpd to a newer version can lead to unexpected results.

You need to configure Berkeley DB with the `-with-uniquename` configure switch to avoid these kind of problems. This insures your new version will not interfere with another installation of Berkeley DB on your system.

```
root# cd build_unix
root# ../dist/configure --prefix=/install/path --with-uniquename
root# make
root# make install
```

If you select an install path other than `/usr/local`, you will have to configure your linker to look for libraries in this directory.

On many operating systems, this is done by adding a new entry to `/etc/ld.so.conf`.

```
root# echo /install/path/lib >/etc/ld.so.conf
root# ldconfig
```

You should now continue with building Netatalk.

Chapter 3

Setting up Netatalk

3.1 Appletalk

AppleTalk, the network protocol family founded by Apple, contains different protocols for different uses (address resolution, address/name mapping, service location, establishing connections, and the like)

A complete overview can be found inside the developer documentation

3.1.1 To use AppleTalk or not

You'll need the AppleTalk support built into netatalk in case you want to provide printing services via PAP by papd(8) or file services via AppleTalk via afpd(8) for older AFP clients not capable of using AFP over TCP. You'll need it also, if you want to use the deprecated AppleTalk-based timeserver timelord(8) for older Mac clients

But even if you don't need PAP or AFP over AppleTalk, you might consider using AppleTalk for service propagation/location, having the ease of use for your network clients in mind. The Apple engineers implemented a way to easily locate an AFP server via AppleTalk but establishing the AFP connection itself via AFP over TCP (see the developer documentation for details on this cool feature, too)

To use the different base AppleTalk protocols with netatalk, one has to use atalkd(8). It can also be used as an AppleTalk router to connect different independent network segments to each other.

To use AppleTalk/atalkd, your system has to have kernel support for AppleTalk. On some systems, that netatalk supports, this isn't currently true (notably True64 Unix) so you can use only netatalk services that do not rely on AppleTalk (which means "AFP over TCP" and requires the -noddp switch in afpd.conf)

3.1.2 No AppleTalk routing

This is the most simple form, you can use AppleTalk with netatalk. In case, you have only one network interface up and running, you haven't to deal with atalkd's config at all: atalkd will use AppleTalk's self-configuration features to get an AppleTalk address and to register itself in the network automagically.

In case, you have more than one active network interface, you have to make a decision:

- Using only one interface: You've to add the interface's name (en1, le0, eth2, ... for example) to atalkd.conf on a single line. Do only list *one* here.

Example 3.1.1: atalkd.conf containing one entry

```
eth0
```

Appletalk networking should be enabled on eth0 interface. All the necessary configuration will be fetched from the network

After startup, atalkd will add the real settings (address and network and eventually a zone) to atalkd.conf on its own

Example 3.1.2: atalkd.conf containing one entry after atalkd started

```
eth0 -phase 2 -net 0-65534 -addr 65280.166
```

atalkd filled in the AppleTalk settings that apply to this network segment. A netrange of 0-65534 is a sign that there is no AppleTalk router present, so atalkd will fetch an address that matches the following criteria: netrange from inside the so called "startup range" 65280-65533 and a node address between 142 and 255

- When using several interfaces you have to add them line by line following the "-dontroute" switch in atalkd.conf

Example 3.1.3: atalkd.conf containing several entries with the -dontroute option

```
eth0 -dontroute
eth1 -dontroute
eth2 -dontroute
```

Appletalk networking should be enabled on all three interfaces but no routing should be done between the different segments. Again, all the necessary configuration will be fetched from the connected networks

Example 3.1.4: atalkd.conf containing several entries with the -dontroute option after atalkd started

```
eth0 -dontroute -phase 2 -net 0-65534 -addr 65280.152
eth1 -dontroute -phase 2 -net 0-65534 -addr 65280.208
eth2 -dontroute -phase 2 -net 1-1000 -addr 10.142 -zone "Printers"
```

On eth0 and eth1 no other routers are present, so atalkd chooses an address from within the startup range. But on eth2 there lives an already connected AppleTalk router, publishing one zone called "Printers" and forcing clients to assign theirselves an address in a netrange between 1 and 1000

In this case atalkd will handle each interface as it would be the only active one. This can have some side effects when it comes to the point where AFP clients want to do the magic switch from AppleTalk to TCP, so use this with caution

In case, you have more than one active network interface and do not take special precautions as outlined above, then autoconfiguration of the interfaces might fail in a situation where one of your network interfaces is connected to a network where *no* other active AppleTalk router is present and supplies appropriate routing settings.

For further information see atalkd.conf(5) and the developer documentation

3.1.3 atalkd acting as an AppleTalk router

There exist several types of AppleTalk routers: seed, non-seed and so called soft-seed routers.

- A seed router has its own configuration and publishes this into the network segments it is configured for
- A non-seed router needs a seed router on the interface to which it is connected to learn the network configuration. So this type of AppleTalk router can work completely without manual configuration.
- A so called soft-seed router is exactly the same as a non-seed router except the fact, that it can also remember the configuration of a seed router and act as a replacement in case, the real seed router disappears from the net.

Netatalk's atalkd can act as both a seed and a soft-seed router, even in a mixed mode, where it acts on one interface in this way and on the other in another.

If you leave your atalkd.conf completely empty or simply add all active interfaces line by line without using seed settings (atalkd will act identically in both cases) then atalkd is forced to act as a soft-seed router on each interface, so it will fail on the first interface, where no seed router is accessible to fetch routing information from.

In this case other services, that depend on atalkd, might also fail.

So you should have atalkd act as a seed router on one or all active interfaces. A seed router has to supply informations about:

- The specific netrange on this segment
- The own AppleTalk address
- The zones (one to many) available in this segment
- The so called "default zone" for this segment

Unless you are the network admin yourself, consider asking her/him before changing anything related to AppleTalk routing, as changing these settings might have side effects for all of your AppleTalk network clients

In an AppleTalk network netranges have to be unique and must not overlap each other. Fortunately netatalk's atalkd is polite enough to check whether your settings are in conflict with already existing ones on the net. In such a case it simply discards your settings and tries to adapt the already established ones on the net (if in doubt, always check syslog for details).

Netranges, you can use, include pretty small ones, eg. 42-42, to very large ones, eg. 1-65279 - the latter one representing the maximum. In routed environments you can use any numbers in the range between 1 and 65279 unless they do not overlap with settings of other connected subnets.

The own AppleTalk address consists of a net part and a node part (the former 16 bit, the latter 8 bit, for example 12057.143). Apple recommends using node addresses of 128 or above for servers, letting client Macs assign themselves an address faster (as they will primarily search for a node address within 1-127 in the supplied netrange). As we don't want to get in conflict with Apple servers, we prefer using node addresses of 142 or above.

AppleTalk zones have *nothing* to do with physical networks. They're just a hint for your client's convenience, letting them locate network ressources in a more comfortable/faster way. You can either use one zone name accross multiple physical segments as well as more than one zone name on a single segment (and various combinations of this)

So all you have to do is to *draw a network chart* containing the physical segments, the netranges you want to assign to each one, the zone names you want to publish in which segments and the default zone per segment (this is always the first zone name, you supply with the "-zone" switch in atalkd.conf)

Given, you finished the steps outlined above, you might want to edit atalkd.conf to fit your needs.

You'll have to set the following options in atalkd.conf:

- -net (use reasonable values between 1-65279 for each interface)

  In case, this value is suppressed but -addr is present, the netrange from this specific address will be used

- -addr (the net part must match the -net settings if present, the node address should be between 142 and 255)

- -zone (can be used multiple times in one single line, the first entry is the default zone)

Note that you are able to set up "zone mapping", that means publishing exactly the same zone name on all AppleTalk segments, as well as providing more than one single zone name per interface. Dumb AppleTalk devices, like LaserWriters, will always register themselves in the default zone (the first zone entry you use in atalkd.conf per interface), more intelligent ones will have the ability to choose one specific zone via a user interface.

Example 3.1.5: atalkd.conf making netatalk a seed router on two interfaces

```
eth0 -seed -phase 2 -net 1-1000 -addr 1000.142 -zone "Printers" -zone "Spoolers"
eth1 -seed -phase 2 -net 1001-2000 -addr 2000.142 -zone "Macs" -zone "Servers"
```

The settings for eth0 force AppleTalk devices within the connected network to assign theirselves an address in the netrange 1-1000. Two zone names are published into this segment, "Printers" being the so called "standard zone", forcing dumb AppleTalk devices like Laserprinters to show up automatically into this zone. AppleTalk printer queues supplied by netatalk's papd can be registered into the zone "Spoolers" simply by adjusting the settings in papd.conf(5). On eth1 we use the different and non-overlapping netrange 1001-2000, set the default zone to "Macs" and publish a fourth zone name "Servers".

Example 3.1.6: atalkd.conf configured for "zone mapping"

```
eth0 -seed -phase 2 -net 1-1000 -addr 1000.142 -zone "foo"
lo0 -phase 1 -net 1 -addr 1.142 -zone "foo"
```

We use the same network settings as in the example above but let atalkd publish the same zone name on both segments. As the same zone name will be used on all segments of the AppleTalk network no zone names will show up at all... but AppleTalk routing will still be active. In this case, we connect a so called "non-extended" LocalTalk network (phase 1) to an EtherTalk "extended" network (phase 2) transparently.

Example 3.1.7: atalkd.conf for a soft-seed router configuration

```
eth0
eth1
eth2
```

As we have more than one interface, atalkd will try to act as an AppleTalk router between both segments. As we don't supply any network configuration on our own we depend on the availability of seed routers in every connected segment. If only one segment is without such an available seed router the whole thing will fail.

Example 3.1.8: atalkd.conf for a soft-seed router configuration after atalkd started

```
eth0 -phase 2 -net 10-10 -addr 10.166 -zone "Parking"
eth1 -phase 2 -net 10000-11000 -addr 10324.151 -zone "No Parking" -zone "Parking"
eth2 -phase 2 -net 65279-65279 -addr 65279.142 -zone "Parking" -zone "No Parking"
```

In this case, active seed routers are present in all three connected networks, so atalkd was able to fetch the network configuration from them and, since the settings do not conflict, act as a soft-seed router from now on between the segments. So even in case, all of the three seed routers would disappear from the net, atalkd would still supply the connected network with the network configuration once learned from them. Only in case, atalkd would be restarted afterwards, the routing information will be lost (as we're not acting as seed router).

Example 3.1.9: atalkd.conf ready for mixed seed/soft-seed mode

```
eth0
eth1 -seed -phase 2 -net 99-100 -addr 99.200 -zone "Testing"
```

In case in the network connected to eth0 lives no active seed router or one with a mismatching configuration (eg. an overlapping netrange of 1-200) atalkd will fail. Otherwise it will fetch the configuration from this machine and will route between eth0 and eth1, on the latter acting as a seed router itself.

By the way: it is perfectly legal to have more than one seed router connected to a network segment. But in this case, you should take care that the configuration of all connected routers is exactly the same regarding netranges, published zone names and also the "standard zone" per segment

3.2 File Services

Netatalk supplies two different transport protocols for AFP services and both can run at the same time. Classic AFP over AppleTalk requires the **afpd** and **atalkd** daemons. AFP over IP only requires **afpd**.

3.2.1 Setting up the AFP file server

AFP (the Apple Filing Protocol) is the protocol Apple Macintoshs use for file services. The protocol has evolved over the years, at the time of this writing 7 different "versions" exist. The latest changes to the protocol, called "AFP 3.1", were added with the release of Panther (Mac OS X 10.3).

AFP3 brought some big changes. For the first time, AppleShare Clients can use filenames up to 255 characters (actually 255 bytes leading to 85-255 chars depending on the glyphs used), UTF-8 is used on the wire and large files (>4GB) are supported.

The afpd deamon offers the fileservices to Apple Clients. It's configured using the `afpd.conf` and the `AppleVolumes.*` files.

3.2.1.1 afpd.conf

afpd.conf is the configuration file used by afpd to determine the behavior and configuration of the different virtual file servers that it provides. Any line not prefixed with '#' is interpreted.

If afpd switches set on the command line are in conflict with afpd.conf settings, the latter will have higher priority

Format: - [options] to specify options for the default server and/or "Server name" [options] to specify an additional server

Leaving the afpd.conf file empty equals to the following configuration:

```
- -transall -uamlist uams_guest.so,uams_clrtxt.so,uams_dhx.so -nosavepassword
```

For a more detailed explanation of the available options, please refer to the afpd.conf(5) man page.

3.2.1.2 AppleVolumes.default

The AppleVolumes.default file is used to define volumes that will by default be shown to all users, including users logged in as guest. A volume will not be presented in the chooser, if the user has no read access to the specified volume path.

You can limit access to a specific volume by using the `allow` and `deny` options.

For a more detailed explanation of the available options, please refer to the AppleVolumes.default(5) man page.

3.2.1.3 CNID backends

Unlike other protocols like smb or nfs, the AFP protocol mostly refers to files and directories by ID and not by a path. A typical AFP request uses a directory ID and a filename, something like "server, please open the file named 'Test' in the directory with id 167". For example "Aliases" on the Mac basically work by ID (with a fallback to the absolute path in more recent AFP clients. But this applies only to Finder, not to applications).

Every file in an AFP volume has to have a unique file ID, IDs must, according to the specs, never be reused, and IDs are 32 bit numbers (Directory IDs use the same ID pool). So, after ~4 billion files/folders have been written to an AFP volume, the ID pool is depleted and no new file can be written to the volume. No whining please :-)

Netatalk needs to map IDs to files and folders in the host filesystem. To achieve this, several different CNID backends are available. A CNID backend is basically a database storing ID <-> name mappings.

In the past, many users used the so called "last" CNID scheme. However, this scheme has some serious drawbacks, as it is based on the device and inode of a file. Therefore, IDs will be eventually be reused and you can get duplicate IDs as well.

The CNID Databases are by default located in the .AppleDB folder in every afpd volume root. With the new ADv2 format, afpd stores the files/directories ID in the corresponding .AppleDouble file as well.

NOTE

There are some CNID related things you should keep in mind when working with netatalk:

- Don't use unix symlinks. Just don't. With a symlink a file/directory "exists" twice, something AFP doesn't allow. There's currently no way this can be resolved, as we either end up with two file/dirs having the same id, or a file having two parents. If you still insist on using them, be aware you're *heavily* violating the specs. You have been warned...
- Don't nest volumes.
- CNID backends are databases, so they turn afpd into a file server/database mix. Keep this in mind, killing an afpd process with **kill -9** will likely leave the database unusable.
- If there's no more space on the filesystem left, the database will get corrupted. You can work around this by either using the -dbpath option and put the database files into another location or, if you use quotas, make sure the .AppleDB folder is owned by a user/group without a quota.
- Be careful with CNID databases for volumes that are mounted via NFS. That is a pretty audacious decision to make anyway, but putting a database there as well is really asking for trouble, i.e. database corruption. Use the dbpath: directive in the `AppleVolumes.*` configuration files to put the databases onto a local disk if you must use NFS mounted volumes.

cdb

The "concurrent database" backend is based on sleepycat's Berkeley DB. With this backend several afpd deamons access the CNID database directly. Berkeley DB locking is used to synchronize access, if more than one afpd process is active for a volume. The drawback is, that the crash of a single afpd process might corrupt the database.

dbd

Access to the CNID database is restricted to the cnid_dbd daemon process. afpd processes communicate with the daemon for database reads and updates. If built with Berkeley DB transactions the probability for database corruption is practically zero, but performance can be slower than with cdb. As a database process gets spawned for each volume, you're probably better off using cdb for sharing home directories for a larger number of users.

last

The last backend is a semi persistent backend. ID's will be reused and, what is much worse, you can get duplicate IDs. You should use it for sharing cdroms only, *don't* use it for sharing normal volumes.

3.2.1.4 Charsets/Unicode

Why Unicode?

Internally computers don't know anything about characters and texts, they only know numbers. Therefore, each letter is assigned a number. A character set, often referred to as *charset* or *codepage*, defines the mappings

between numbers and letters.

If two or more computer systems need to communicate with each other, the have to use the same charater set. In the 1960s the ASCII (American Standard Code for Information Interchange) character set was defined by the American Standards Association. The original form of ASCII represented 128 characters, more than enough to cover the English alphabet and numerals. Up to date, ASCII has been the normative character scheme used by computers.

Later versions defined 256 characters to produce a more international fluency and to include some slightly esoteric graphical characters. Using this mode of encoding each character takes exactly one byte. Obviously, 256 characters still wasn't enough to map all the characters used in the various languages into one character set.

As a result localized character sets were defined later, e.g the ISO-8859 character sets. Most operating system vendors introduced their own characters sets to satisfy their needs, e.g. IBM defined the *codepage 437 (DOSLatinUS)*, Apple introduced the *MacRoman* codepage and so on. The characters that were assigned number larger than 127 were referred to as *extended* characters. These character sets conflict with another, as they use the same number for different characters, or vice versa.

Almost all of those characters sets defined 256 characters, where the first 128 (0-127) character mappings are identical to ASCII. As a result, communication between systems using different codepages was effectivly limited to the ASCII charset.

To solve this problem new, larger character sets were defined. To make room for more character mappings, these character sets use at least 2 bytes to store a character. They are therefore referred to as *multibyte* character sets.

One standardized multibyte charset encoding scheme is known as unicode <http://www.unicode.org/>. A big advantage of using a multibyte charset is that you only need one. There is no need to make sure two computers use the same charset when they are communicating.

character sets used by Apple

In the past, Apple clients used single-byte charsets to communicate over the network. Over the years Apple defined a number of codepages, western users will most likely be using the *MacRoman* codepage.

Codepages defined by Apple include:

- MacArabic, MacFarsi
- MacCentralEurope
- MacChineseSimple
- MacChineseTraditional
- MacCroation
- MacCyrillic
- MacDevanagari
- MacGreek
- MacHebrew
- MacIcelandic
- MacKorean
- MacJapanese
- MacRoman

- MacRomanian
- MacThai
- MacTurkish

Starting with Mac OS X and AFP3, UTF-8 <http://www.utf-8.com/> is used. UTF-8 encodes Unicode characters in an ASCII compatible way, each Unicode character is encoded into 1-6 ASCII characters. UTF-8 is therefore not really a charset itself, it's an encoding of the Unicode charset.

To complicate things, Unicode defines several *normalization* <*http://www.unicode.org/reports/tr15/index.html*> forms. While samba <http://www.samba.org> uses *precomposed* Unicode, which most Unix tools prefer as well, Apple decided to use the *decomposed* normalization.

For example lets take the German character 'ä'. Using the precomposed normalization, Unicode maps this character to 0xE4. In decomposed normalization, 'ä' is actually mapped to two characters, 0x61 and 0x308. 0x61 is the mapping for an 'a', 0x308 is the mapping for a *COMBINING DIAERESIS*.

Netatalk referres to precomposed UTF-8 as *UTF8* and to decomposed UTF-8 as *UTF8-MAC*.

afpd and character sets

To support new AFP 3.x and older AFP 2.x clients at the same time, afpd needs to be able to convert between the various charsets used. AFP 3.x clients always use UTF-8, AFP 2.2 clients use one of the Apple codepages.

At the time of this writing, netatalk supports the following Apple codepages:

- MAC_CENTRALEUROPE
- MAC_CYRILLIC
- MAC_HEBREW
- MAC_ROMAN
- MAC_TURKISH

afpd handles three different character set options:

unixcodepage This is the codepage used internally by your operating system. If not specified and your system support Unix locales, afpd tries to detect the codepage, otherwise it defaults to ASCII. afpd uses this codepage to read its configuration files, so you can use extended characters for volume names, login messages, etc. see afpd.conf(5)

maccodepage As already mentioned, older MacOS clients (up to AFP 2.2) use codepages to communicate with afpd. However, there is no support for negotiating the codepage used by the client in the AFP protocol. If not specified otherwise, afpd assumes the *MacRoman* codepage is used. In case you're clients use another codepage, e.g. *MacCyrillic*, you'll **have** to explicitly configure this. see afpd.conf(5)

volcharset This defines the charset afpd should use for filenames on disk. The default is UTF8. If you have iconv <http://www.gnu.org/software/libiconv/> installed, you can use any iconv provided charset as well.

afpd needs a way to preserve extended macintosh characters, or characters illegal in unix filenames, when saving files on a unix filesystem. Earlier versions used the the so called CAP encoding. An extended character (>0x7F) would be converted to a :xx hex sequence, e.g. the Apple Logo (MacRoman: 0XF0) was saved as :f0. Some special characters will be converted as to :xx notation as well. '/' will be encoded to :2f, if -usedots is not specified, a leading dot '.' will be encoded as :2e. Even though this version

now uses UTF-8 as the default encoding for filenames, special characters, like '/' and a leading '.' will still be CAP style encoded . For western users another useful setting could be *-volcharset ISO-8859-15*.

If a character cannot be converted from the mac codepage to the selected volcharset, afpd will save it as a CAP encoded character. For AFP3 clients, afpd will convert the UTF-8 character to *maccodepage* first. If this conversion fails, you'll receive a -50 error on the mac. *Note*: Whenever you can, please stick with the default UTF-8 volume format. see AppleVolumes.default(5)

3.3 Printing

Netatalk can act as both a PAP client to access AppleTalk-capable printers and a PAP server. The former by using the **pap**(1) utility and the latter by starting the **papd**(8) service.

The "Printer Access Protocol" as part of the AppleTalk protocol suite is a fully 8 bit aware and bidirectional printing protocol, developed by Apple in 1985. *8 bit aware* means that the whole set of bytes can be used for printing (binary encoding). This has been a great advantage compared with other protocols like Adobe's Standard Protocol to drive serial and parallel PostScript printers (compare with Adobe TechNote 5009 <http://partners.adobe.com/asn/tech/ps/specifications.jsp>) or LPR which made only use of the lower 128 bytes for printing because the 8th bit has been reserved for control codes.

Bidirectional means that printing source (usually a Macintosh computer) and destination (a printer or spooler implementation) communicate about both destination's capabilities via feature queries (compare with Adobe TechNote 5133 <http://partners.adobe.com/asn/tech/ps/archive.jsp>) and device status. This allows the LaserWriter driver on the Macintosh to generate appropriate device specific PostScript code (color or b/w, only embetting needed fonts, and so on) on the one hand and notifications about the printing process or problems (paper jam for example) on the other.

3.3.1 Setting up the PAP print server

Netatalk's **papd** is able to provide AppleTalk printing services for Macintoshes or, to be more precise, PAP clients in general. Netatalk does not contain a full-blown spooler implementation itself, papd only handles the bidirectional communication and submittance of printjobs from PAP clients. So normally one needs to integrate papd with a Unix printing system like eg. classic SysV lpd, BSD lpr, LPRng, CUPS or the like.

Since it is so important to answer the client's feature queries correctly, how does papd achieve this? By parsing the relevant keywords of the assigned PPD file. That said, it's always necessary to carefully choose the right PPD at the server's side.

Netatalk had built-in support for System V lpd printing in a way that papd saved the printed job directly into the spooldir and calls **lpd** afterwards, to pick up the file and do the rest. Due to incompatibilities with many lpd implementations the normal behaviour was to print directly into a pipe instead of specifying a printer by name and using lpd interaction. With Netatalk 2.0 another alternative has been implemented: directly integrating with CUPS (Note: when CUPS support is compiled in, then the SysV lpd support doesn't work at all). Detailed examples can be found in the papd.conf(5) manual page.

3.3.1.1 Integrating papd with SysV lpd

Unless CUPS support has been compiled in (which is default from Netatalk 2.0 on) one simply defines the lpd queue in question by setting the `pr` parameter to the queue name, in the following example "ps". If no `pr` parameter is set, the default printer will be used.

3.3.1.2 Using pipes with papd

An alternative to the technique outlined above is to direct papd's output via a pipe into another program. Using this mechanism almost all printing systems can be driven. Netatalk supplies three "wildcards" that get substituted with values of the already printed job:

%F will be substituted with the contents of the *%%For:* comment in the PostScript stream.

%U If authenticated printing has been enabled then this will be substituted with the user name of the print-job's originator.

%J will be substituted with the contents of the *%%Title:* comment of the PostScript stream.

Using these wildcards, one can pass those parameters directly to programs or implement small wrapper scripts to call the printing system in question.

3.3.1.3 Using direct CUPS support

Starting with Netatalk 2.0 direct CUPS integration is available. In this case, defining only a queue name as `pr` parameter won't invoke the SysV lpd daemon but uses CUPS instead. Unless a specific PPD has been assigned using the `pd` switch, the PPD configured in CUPS will be used by **papd**, too.

There exists one special share named "cupsautoadd". If this is present as the first entry of papd.conf then all available CUPS queues will be served automagically using the parameters assigned to this global share. But subsequent printer definitions can be used to override these global settings for individual spoolers.

3.3.2 Using AppleTalk printers

Netatalk's **papstatus**(8) can be used to query AppleTalk printers, **pap**(1) to print to them. With **psf**(8) there exists a lpd filter program suitable for converting other formats (like text) to PostScript output, do page accounting and eventually change the page order using **psorder**(1). But these days, modern printing systems like CUPS can do the latter tasks for themselves in a more realiable way.

pap can be used stand-alone or as part of an output filter or a CUPS backend (which is the preferred method since one does not have to deal with all the options).

Example 3.3.1: pap printing to a PostScript LaserWriter

```
pap -p"ColorLaserWriter 16/600@*" /usr/share/doc/gs/examples/tiger.ps
```

The file `/usr/share/doc/gs/examples/tiger.ps` is sent to a printer called "ColorLaserWriter 16/600" in the standard zone "*". The device type is "LaserWriter" (can be suppressed since it is the default).

Example 3.3.2: pap printing to a non-PostScript printer

```
gs -q -dNOPAUSE -sDEVICE=cdjcolor -sOutputFile=- test.ps | pap -E
```

GhostScript is used to convert a PostScript job to PCL3 output suitable for a Color DeskWriter. Since no file has been supplied on the command line, **pap** reads the data from stdin. The printer's address will be read from the `.paprc` file in the same directory, **pap** will be called (in our example simply containing "Color DeskWriter:DeskWriter@Printers"). The `-E` switch forces **pap** to not wait for an EOF from the printer.

3.4 Time Services

3.4.1 Using Netatalk as a time server for Macintoshs

"timelord", an AppleTalk based time server, is deprecated these days. Use NTP instead.

3.5 Starting and stopping Netatalk

The Netatalk distribution comes with several operating system specific startup script templates that are taylored according to the options given to the "configure" script before compiling. Currently, templates are provided for Netbsd, Bsd, Redhat, Suse and True64. You can select to install the generated startup script(s) by specifying a system type to "configure". To automatically install startup scripts for e.g. the Suse Linux distribution try to give the –enable-suse option to "configure". Some of the scripts can be further parametrized by the configuration file netatalk.conf (described in the netatalk.conf(5) manual page), some obtain that information in another, operating system specific way (like Netbsd).

Since new releases of Linux distributions appear all the time and the startup procedure for the other systems mentioned above might change as well, it is probably a good idea to not blindly install a startup script but to look at it first to see if it will work on your system. If you use Netatalk as part of a fixed setup, like a Linux distribution, an RPM or a BSD package, things will probably have been arranged properly for you. The following therefore applies mostly for people who have compiled Netatalk themselves.

The following daemons need to be started by whatever startup script mechanism is used:

- atalkd (if you use the AppleTalk protocol)
- afpd
- cnid_metad (if the dbd CNID backend is used)
- papd (if you want to provide print services via AppleTalk)
- timelord (for old style time synchronisation via AppleTalk)

Additionally, make sure that the various configuration files (afpd.conf, AppleVolumes.default, papd.conf etc.) are in the right place and that netatalk.conf (if used) contains the right entries. If you want e.g. papd to be started using this mechanism, set the environment variable "PAPD_RUN" to "yes" in netatalk.conf. See the manual pages for details.

Chapter 4

Upgrading from a previous version of Netatalk

4.1 Overview

Version 2.0 of the Netatalk suite includes significant changes and enhancements in functionality compared to previous versions. AFP 3.x is now supported which allows UTF-8 encoded filenames of up to 255 bytes (85-255 chars) in length amongst other things. The Catalogue Node ID (CNID) subsystem has been reworked as well and should now be much more robust. For an overview of what CNIDs are and why you need them please see the CNID section in the manual.

The downside of these enhancements is that upgrading to Netatalk 2.0 is not a process that can be easily automated. Too many factors depend on site specific configuration and administrators have to make choices that suit their requirements. This document attempts to clarify the issues and outline the steps that need to be taken for a successful upgrade. As usual, the first of these steps should be to make a complete backup of all volumes and home directories that were in use with Netatalk before. Afterwards, you'll have to decide

1. what encoding to use for filenames in the future and how to convert existing filenames
2. what storage scheme to use for CNIDs and maybe convert an existing database to that scheme

The following two sections deal with each of these areas in turn

4.2 Volumes and filenames

Previous Netatalk versions saved filenames in the so called *CAP* encoding by default. Alternatively, there was the NLS system, that allowed you to convert filenames to other codepages, like ISO-8859-1.

For Netatalk 2.0 the charset conversion routines had to be completely rewritten to support AFP 3.x. For more indepth information on character sets please read the Unicode/charsets section in the manual.

As a consequence, Netatalk 2.0 now stores filenames in UTF-8 by default. Additionally you **have** to specify a maccodepage in afpd.conf, if your Mac clients are not using *MacRoman*.

The format of the metadata files stored in the .AppleDouble folders has changed from AppleDouble v1 to AppleDouble v2. Netatalk 2.0 is still able to use AD1 files, if configured. Otherwise ADv1 files will silently be updated to the new ADv2 format, which will prevent you from using this volume with 1.x again.

WARNING

Do not share a 1.x volume with Netatalk 2.0 without setting the proper options!

NOTE

You should consider 'upgrading' your volumes using the new defaults UTF-8 and AppleDouble v2, even if this is a time consuming process. AFP 3.x uses UTF-8 and it is impossible to fully map UTF-8 to any of the old volume formats.

4.2.1 How to upgrade a volume to 2.0

To convert the 1.x CAP or NLS encoded volumes on the server, we provide the uniconv(1) utility. Please see the man page for details.

Another option to perform an upgrade, is to copy all files using a Mac client. Either copy the volume to a Mac while you are still running 1.6, then install 2.0 and copy the data back to a fresh share, or try to set up the volume with the compatibility options described below and do a share to share copy.

4.2.2 How to use a 1.x CAP encoded volume with 2.0

Using a 1.x CAP encoded volume is still possible with Netatalk 2.0. To work properly, the following options need to be set, matching your 1.x setup:

afpd.conf:

- maccodepage

AppleVolumes.default:

- volcharset
- adouble

You have to make sure `maccodepage` matches your Apple clients codepage. For western users the default *Mac_Roman* should be fine.

Set `volcharset` to *ASCII*.

Set `adouble:v1`, this will make sure the metadata files will not be changed to AppleDouble v2. If you do not set this option, it will not be possible to use the volume with Netatalk 1.x anymore.

Example:

`afpd.conf`:

```
- -transall -maccodepage:MAC_CENTRALEUROPE
```

`AppleVolumes.default`:

```
/path/to/share "1.x Volume" adouble:v1 volcharset:ASCII
```

4.2.3 How to use a 1.x NLS volume with 2.0

Whether you can still use an 1.x NLS encoded volume with Netatalk 2.0 mainly depends on which NLS setting you used with 1.x.

Make sure you set the correct `maccodepage` in `afpd.conf` !

maccode.iso8859-1 Use the following settings in `AppleVolumes.default`:

```
/path/to/share "1.x Volume" adouble:v1 volcharset:ISO-8859-1
```

maccode.iso8859-1.adapted Sorry, you're out of luck. This NLS contains a non standard mapping and is not supported by afpd anymore. You'll have to convert the volume to a supported encoding.

maccode.437 Using the following settings in `AppleVolumes.default` might work, but is untested:

```
/path/to/share "1.x Volume" adouble:v1 volcharset:CP437
```

maccode.850 Using the following settings in `AppleVolumes.default` might work, but is untested:

```
/path/to/share "1.x Volume" adouble:v1 volcharset:CP850
```

maccode.koi8-r Using the following settings in `AppleVolumes.default` might work, but is untested:

```
/path/to/share "1.x Volume" adouble:v1 volcharset:KOI8-R
```

NOTE

All of the above require iconv <http://www.gnu.org/software/libiconv/> to be installed and to supply the volcharset codepage!

4.3 Choosing a CNID storage scheme

Previous versions of Netatalk allocated CNIDs either on the fly or CNIDs were recorded in a persistent database. "On the fly methods" work by either generating a CNID from the device and inode number or simply by using a counter that is increased by one on each access to a file or directory from the client. The counter only lasts for the lifetime of an afpd daemon process and inode numbers are resused for a different file once the original file has been deleted. These methods therefore violate a fundamental assumption: A CNID once assigned must never be reused for the lifetime of a volume. Netatalk 2.0 supports one "On the fly scheme" called *last*. It computes CNIDs for files from device and inode of the file and uses a counter for directories. You should think twice about using it in production. Depending on your needs and the semantics

of the underlying file system it might be OK on read only volumes, but even there we are not certain if OS X clients will work properly.

That leaves the CNID schemes that use persistent storage for CNIDs. Netatalk 2.0 supports two: *cdb* and *dbd*. Both are based on the Berkeley DB database library as before. One difference is, though, that you are not restricted to using a single scheme for all of your volumes that has to be determined at compile time. The CNID scheme (also called a "CNID backend") is now a runtime option for a volume. That means that you can make the choice per volume based on your requirements. Here are the properties as well as the advantages and disadvantages of the three supported schemes:

1. **last:** See above. Avoid, if at all possible.
2. **cdb:**Roughly analogous to the Netatalk 1.6.x versions with what was called then the "DID scheme" option set to "cnid" and the "CNID with Concurrent Data Store" option set to "yes". Access to the CNID database for a volume happens directly from the Netatalk afpd daemons. A Berkeley DB locking scheme (the "Concurrent Data Store" bit) is used to avoid database inconsistencies. Robustness is much improved compared to previous releases. The CNID database can only become corrupted if an afpd daemon crashes unexpectedly, is killed by the administrator or the whole machine crashes.
3. **dbd:** There is only a single daemon that accesses the CNID database for a given volume. Any afpd process that wishes to retrieve or update CNIDs for that volume needs to do it via the daemon. The CNID can database be (this is a compile time option) updated under Berkeley DB transactional protection. This design combined with the transactional updates makes the CNID database crashproof: Any of the participating afpd daemons, the database daemon itself or the whole machine can crash and the CNID database should still be in a consistent state. The downside to this is that the speed of updates and retrieval is slower than with the *cdb* scheme. If this is a problem, you might want to disable transactions at Netatalk compile time (currently, the default is to compile without transactions anyway). That will give you safety against afpd crashing, but not if the machine goes down unexpectedly. Also, have a look at the nosync option documented in the cnid_dbd manual page.

It is also possible to switch between *cdb* and *dbd* for a given volume, since they use the same database format. You just have to shut down all processes accessing the database cleanly, make the necessary configuration changes and restart.

Note that the *dbd* backend needs an auxiliary daemon, called cnid_metad, to work. It should be started together with afpd. If the *dbd* backend is compiled into afpd (the default), this should happen automatically. If you cannot find it in the process list even though the *dbd* backend is used please check for errors in the startup scripts.

If you compile Netatalk 2.0 yourself and invoke configure –help, you'll notice that there are in fact more CNID backends to chose from. Don't use any of them. They are either broken or incomplete. Some of them might turn into something useful in the future.

4.3.1 How to upgrade if no persistent CNID storage was used

That is easy. Just pick a CNID backend from above, configure it properly in afpd.conf and the AppleVolumes file and start up the necessary Netatalk processes. The databases will be automatically created in a subdirectory .AppleDB of the volume in question.

4.3.2 How to upgrade if a persistent CNID storage scheme was used

In that case the CNID databases need to be upgraded. A script called cnid2_create that comes with Netatalk 2.0 does most of the work. The steps you have to take depend on what version of Berkeley DB is installed on your system. If you already use one of the supported versions of Berkeley DB (4.1.25 or 4.2.52) for your old Netatalk installation and plan to use it for Netatalk 2.0 as well just use the db_dump and db_load utilities that came with it as indicated below. Otherwise it is probably best to have the old and the new (to be used with

Netatalk 2.0) version of Berkeley DB installed side by side until you have finished the upgrade. The reason for this is that we will dump out the old databases with the currently installed version of Berkeley DB in ASCII format and reload them with the new version. This avoids any incompatibility problems between Berkeley DB releases with respect to the on-disk format.

For each volume to be upgraded, follow these steps

- Stop all afpd daemons accessing the volume.
- Change to the database directory for that volume, most likely the .AppleDB subdirectory of the volume toplevel directory in question.
- Dump the contents of cnid.db and didname.db using the old (installed) version of Berkeley DB like this:

  ```
  db_dump -f cnid.dump cnid.db
  db_dump -f didname.dump didname.db
  ```

  Make sure the db_dump utility you are using is the correct (currently used) one. Use the full path to the db_dump executable if in doubt. So if this database was created with Berkeley DB 3.xx installed in /usr/local/db3 use /usr/local/db3/bin/db_dump instead. This will create two files, cnid.dump and didname.dump in the current directory.
- Run the cnid2_create script:

  ```
  /path/to/netatalk/bin/cnid2_create
  ```

  The script assumes that .AppleDB is a subdirectory of the volume directory to be upgraded. If that is not the case give the full path name of the volume as the first argument to cnid2_create. The script will create a file cnid2.dump in ASCII format.
- Remove the old Berkeley DB environment and logfiles (if present):

  ```
  rm __db.* log.*
  ```

- Load cnid2.dump into the new database. You should use the db_load utility of Berkeley DB that will be used with version 2.0 of Netatalk. So if Berkeley DB 4.xx lives in /usr/local/db4 use

  ```
  /usr/local/db4/bin/db_load -f cnid2.dump cnid2.db
  ```

  This will create the new database file, cnid2.db. Remove the old database files cnid.db, didname.db and devino.db. The intermediate files cnid.dump, didname.dump and cnid2.dump can be removed now or at some later time.

If you do not want to have two versions of Berkeley DB installed side by side during the upgrade, you should first dump out the old databases as indicated above for all volumes, upgrade Berkeley DB and then load them with db_load. The cnid2_create script can be run before or after the upgrade. Berkeley DB environment and logfiles should still be removed before running db_load.

4.3.3 How to upgrade if a persistent CNID storage scheme was used, the brute force approach

If you are absolutely sure what you are doing, you can also just clear out all database files from the .AppleDB directories. They will be recreated, but will not contain the same CNIDs as before!! That might lead to all sorts of problems, like aliases not working any more on clients. As I said, make sure you know the consequences and don't mind them.

Chapter 5

Manual Pages

This a collection of the man pages delivered with Netatalk.

5.1 achfile

Name

achfile — change type and/or creator of Apple Macintosh files (netatalk format)

Synopsis

`achfile [ -t type ] [ -c creator ] file...`

DESCRIPTION

achfile changes the Macintosh type and/or creator of the *file* arguments which have a corresponding .AppleDouble file.

OPTIONS

`-t type` change the type.

`-c creator` change the creator.

DIAGNOSTICS

returns exit status 0 if all files changed successfully

SEE ALSO

afile(1), afpd(8)

5.2 acleandir

Name

acleandir — clean up a directory containing netatalk Apple Macintosh files

Synopsis

`acleandir [-rnvi]` ***dirname***

DESCRIPTION

acleandir cleans up the directory *dirname.* By default it simply removes "orphan" AppleDouble files, i.e. those which do not have a corresponding data file.

OPTIONS

`-d` Also remove the .AppleDouble directory if it contains no AppleDouble files after "orphan" removal. This will result in the finder location of *dirname* within its parent being lost.

`-r`, `-R` Recursive. Clean up directories recursively.

`-n` Display the filenames of "orphans" but don't remove any. Display size if "orphan" appears to contain a resource fork.

`-i` Interactive. Prompt for confirmation before a removal. A y in answer confirms that the removal should proceed.

`-v` Verbose. Display the names of all "orphans" and .AppleDouble directories removed. Reports the size if the "orphan" appears to contain a resource fork.

`-a` Aggressive. Remove all AppleDouble files, not just "orphans". Also remove the .AppleDesktop directory if present. Impies `-d` option. Use with caution as the Macintosh type/creator and finder location of all files will be lost and the content of some documents, such as Symantec Projects, will be destroyed.

DIAGNOSTICS

returns exit status 0 unless bad options are provided or a directory is not given on the command line.

SEE ALSO

afile(1), afpd(8)

5.3 aecho

Name

aecho — send AppleTalk Echo Protocol packets to network hosts

Synopsis

aecho [-c*count*] (**address** | **nbpname**)

DESCRIPTION

aecho repeatedly sends an Apple Echo Protocol (AEP) packet to the host specified by the given AppleTalk **address** or **nbpname** and reports whether a reply was received. Requests are sent at the rate of one per second.

address is parsed by atalk_aton(3). **nbpname** is parsed by nbp_name(3). The nbp type defaults to '*Workstation*'.

When **aecho** is terminated, it reports the number of packets sent, the number of responses received, and the percentage of packets lost. If any responses were received, the minimum, average, and maximum round trip times are reported.

EXAMPLE

Check to see if a particular host is up and responding to AEP packets:

```
example% aecho bloodsport
11 bytes from 8195.13: aep_seq=0. time=10. ms
11 bytes from 8195.13: aep_seq=1. time=10. ms
11 bytes from 8195.13: aep_seq=2. time=10. ms
11 bytes from 8195.13: aep_seq=3. time=10. ms
11 bytes from 8195.13: aep_seq=4. time=10. ms
11 bytes from 8195.13: aep_seq=5. time=9. ms
^C
----8195.13 AEP Statistics----
6 packets sent, 6 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 9/9/10
```

OPTIONS

-c*count* Stop after *count* packets.

SEE ALSO

ping(1), atalk_aton(3), nbp_name(3), aep(4), atalkd(8).

5.4 afile

Name

afile — display type and creator of Apple Macintosh files (netatalk format)

Synopsis

`afile [-a] file...`

DESCRIPTION

afile displays the name and Macintosh type and creator of the *file* arguments. Tests whether the file is an AppleDouble header, in which case it checks the corresponding data fork exists, or assumes it is a data fork in which case it looks for the corresponding AppleDouble to find the type/creator information.

afile does not look at any of the extension mapping files such as AppleVolumes.system.

OPTIONS

-a Include directories and data files of unknown type (i.e. without corresponding AppleDouble) in output.

DIAGNOSTICS

returns exit status 0 if all files have a corresponding valid .AppleDouble header or data fork, or 99 for bad command line options. Otherwise it returns the following error code relating to the last invalid file.

1 file doesn't exist

2 file is unreadable

3 file is directory

4 file is AppleDouble without data fork

5 file is AppleDouble with unreadable data fork

6 file is data fork without AppleDouble

7 file is data fork with unreadable AppleDouble

8 file is data fork with short AppleDouble

9 bad magic in AppleDouble

SEE ALSO

achfile(1), afpd(8)

5.5 afpd

Name

afpd — AppleTalk Filing Protocol daemon

Synopsis

`afpd [-duptDTvI] [-f defaultvolumes] [-s systemvolumes] [-n nbpname] [-c maxconnections] [-g guest] [-P pidfile] [-S port] [-L message] [-F config] [-U uamsv] [-m umask]`

Description

afpd provides an AppleTalk Filing Protocol (AFP) interface to the Unix file system. It is normally started at boot time from /etc/rc.

The list of volumes offered to the user is generated from `/etc/netatalk/AppleVolumes.system` and one of `/etc/netatalk/AppleVolumes.default`, `~/AppleVolumes`, or `~/.AppleVolumes`. The `AppleVolumes` files is used to specify volumes to mount and file name extension mappings. It is formatted as follows, one specification per line: pathname [volumename] .extension [type [creator]] If volumename is unspecified, the last component of pathname is used. No two volumes may have the same name. If type is unspecified '????' is used. If creator is unspecified 'UNIX' is used. The extension '.' sets the default creator and type for otherwise untyped Unix files. Blank lines and lines beginning with '#' are ignored.

Options

-d Specifies that the daemon should not fork. If netatalk has been configured with *–enable-debug1*, a trace of all AFP commands will be written to stdout.

-p Prevents clients from saving their passwords. (Equivalent to `-nosavepasswd` in `afpd.conf`.)

-t Allows clients to change their passwords. (Equivalent to `-setpasswd` in `afpd.conf`.)

-D Use DDP (AppleTalk) as transport protocol. (Equivalent to `-ddp` in `afpd.cond`.)

-T Use TCP/IP as transport protocol. (Equivalent to `-tcp` in `afpd.conf`.)

-v Print version information and exit.

-I Use a platform specific icon. (Equivalent to `-icon` in `afpd.conf`.)

-f *defaultvolumes* Specifies that *defaultvolumes* should be read for a list of default volumes to offer, instead of `/etc/netatalk/AppleVolumes.default`.

-s *systemvolumes* Specifies that *systemvolumes* should be read for a list of volume that all users will be offered, instead of `/etc/netatalk/AppleVolumes.system`.

-u Read the user's `AppleVolumes` file first. This option causes volume names in the user's `AppleVolumes` file to override volume names in the system's `AppleVolumes` file. The default is to read the system `AppleVolumes` file first. Note that this option doesn't effect the precendence of filename extension mappings: the user's `AppleVolumes` file always has precedence.

-n *nbpname* Specifies that *nbpname* should be used for NBP registration, instead of the first component of the hostname in the local zone.

-c *maxconnections* Specifies the maximum number of connections to allow for this **afpd**. The default is 20.

-g *guest* Specifies the name of the guest account. The default is *nobody*'.

-P *pidfile* Specifies the file in which **afpd** stores its process id.

-S *port* Specifies the port to register with when doing AFPoverTCP. Defaults to 548. (Equivalent to `-port`in `afpd.conf`.)

-L *message* Specifies the login message that will be sent to clients. (Equivalent to `-loginmsg` in `afpd.conf`.)

-F *configfile* Specifies the configuration file to use. (Defaults to `/etc/netatalk/netatalk/afpd.conf`.)

-U *uams* Comma-separated list of UAMs to use for the authentication process. (Equivalent to `-uamlist` in `afpd.conf`.)

-m *umask* Use this *umask* for the creation of folders in Netatalk.

SIGNALS

Signals that are sent to the main **afpd** process are propagated to the children, so all will be affected.

SIGHUP Sending a **SIGHUP** to afpd will cause it to reload its configuration files.

SIGUSR1 The **afpd** process will send the message "The server is going down for maintenance." to the client and shut itself down in 5 minutes. New connections are not allowed. If this is sent to a child afpd, the other children are not affected. However, the main process will still exit, disabling all new connections.

SIGUSR2 The **afpd** process will look in the message directory configured at build time for a file named message.pid. For each one found, a the contents will be sent as a message to the associated AFP client. The file is removed after the message is sent. This should only be sent to a child **afpd**. Warning: If the –with-message-dir option was not used, this will kill the **afpd** process

To shut down a user's **afpd** process it is recommended that **SIGKILL (-9)** *NOT* be used, except as a last resort, as this may leave the CNID database in an inconsistent state. The safe way to terminate an **afpd** is to send it a **SIGTERM (-15)** signal and wait for it to die on its own.

FILES

`/etc/netatalk/AppleVolumes.default` list of default volumes to mount

`/etc/netatalk/AppleVolumes.system` list of volumes to offer all users

`~/AppleVolumes` user's list of volumes to mount

`/etc/netatalk/netatalk/msg/message.pid` contains messages to be sent to users.

BUGS

SEE ALSO

hosts_access(5), afpd.conf(5), AppleVolumes.default(5), AppleVolumes.system(5).

5.6 afpd.conf

Name

afpd.conf — Configuration file used by afpd(8) to determine the setup of its file sharing services

Description

`/etc/netatalk/afpd.conf` is the configuration file used by **afpd** to determine the behavior and configuration of the different virtual file servers that it provides.

Any line not prefixed with # is interpreted. The configuration lines are composed like: server name [options] If a - is used instead of a server name, the default server is specified. Server names must be quoted if they contain spaces. They must not contain ":" or "@". The path name must be a fully qualified path name, or a path name using either the ~ shell shorthand or any of the substitution variables, which are listed below.

NOTE

Each server has to be configured on a **single** line.

The possible options and their meanings are:

AppleVolumes Files

-defaultvol *[path]* Specifies path to AppleVolumes.default file (default is `/etc/netatalk/AppleVolumes.default`).

-systemvol *[path]* Specifies path to AppleVolumes.system file (default is `/etc/netatalk/AppleVolumes.system`).

-[no]uservol Enables or disables reading of the users' individual volumes file entirely.

-[no]uservolfirst Enables or disables reading of the users' individual volumes file before processing the global `AppleVolumes.default` file.

Authentication Methods

-uamlist *[uams list]* Comma separated list of UAMs. (The default is uams_guest.so,uams_clrtxt.so,uams_dhx.so).

The most commonly used UAMs are:

uams_guest.so allows guest logins

uams_clrtxt.so (uams_pam.so or uams_passwd.so) Allow logins with passwords transmitted in the clear.

uams_randum.so allows Random Number and Two-Way Random Number Exchange for authentication (requires a separate file containing the passwords, either /etc/netatalk/afppasswd file or the one specified via `-passwdfile`. See afppasswd(1) for details

uams_dhx.so (uams_dhx_pam.so or uams_dhx_passwd.so) Allow Diffie-Hellman eXchange (DHX) for authentication.

uam_gss.so Allow Kerberos V for authentication (optional)

-uampath *[path]* Sets the default path for UAMs for this server (default is /etc/netatalk/uams).

-k5keytab *[path]*, -k5service *[service]*, -k5realm *[realm]* These are required if the server supports the Kerberos 5 authentication UAM.

Codepage Options

With OS X Apple introduced the AFP3 protocol. One of the big changes was, that AFP3 uses Unicode names encoded as UTF-8 decomposed. Previous AFP/OS versions used codepages like MacRoman, MacCentralEurope, etc.

To be able to serve AFP3 and older clients at the same time, **afpd** needs to be able to convert between UTF-8 and Mac codepages. Even OS X clients partly still rely on codepages. As there's no way, **afpd** can detect the codepage a pre AFP3 client uses, you have to specify it using the `-maccodepage` option. The default is MacRoman, which should be fine for most western users.

As **afpd** needs to interact with unix operating system as well, it need's to be able to convert from UTF-8/MacCodepage to the unix codepage. By default **afpd** uses the systems LOCALE, or ASCII if your system doesn't support locales. You can set the unix codepage using the `-unixcodepage` option. If you're using extended characters in the configuration files for **afpd**, make sure your terminal matches the `-unixcodepage`.

-unixcodepage [*CODEPAGE*] Specifies the servers unix codepage, e.g. "ISO-8859-15" or "UTF8". This is used to convert strings to/from the systems locale, e.g. for authenthication, server messages and volume names. Defaults to LOCALE if your system supports it, otherwise ASCII will be used.

-maccodepage [*CODEPAGE*] Specifies the mac clients codepage, e.g. "MAC_ROMAN". This is used to convert strings and filenames to the clients codepage for OS9 and Classic, i.e. for authentication and AFP messages (SIGUSR2 messaging). This will also be the default for the volumes maccharset. Defaults to MAC_ROMAN.

Password Options

-loginmaxfail [*number*] Sets the maximum number of failed logins, if supported by the UAM (currently none)

-passwdfile [*path*] Sets the path to the Randnum UAM passwd file for this server (default is /etc/netatalk/afppasswd).

-passwdminlen [*number*] Sets the minimum password length, if supported by the UAM

-[no]savepassword Enables or disables the ability of clients to save passwords locally

-[no]setpassword Enables or disables the ability of clients to change their passwords via chooser or the "connect to server" dialog

Transport Protocols

-[no]ddp Enables or disables AFP-over-Appletalk. If `-proxy` is specified, you must instead use `-uamlist ""` to prevent DDP connections from working.

-[no]tcp Enables or disables AFP-over-TCP

-transall Make both available (default)

Transport Options

-advertise_ssh Allows Mac OS X clients (10.3.3 or above) to automagically establish a tunneled AFP connection through SSH. If this option is set, the server's answers to client's FPGetSrvrInfo requests contain an additional entry. It depends on both client's settings and a correctly configured and running sshd(8) on the server to let things work.

NOTE

Setting this option is not recommended since globally encrypting AFP connections via SSH will increase the server's load significantly. On the other hand, Apple's client side implementation of this feature in MacOS X versions prior to 10.3.4 contained a security flaw

-ddpaddr *[ddp address]* Specifies the DDP address of the server. The default is to auto-assign an address (0.0). This is only useful if you are running AppleTalk on more than one interface.

-fqdn *[name:port]* Specifies a fully-qualified domain name, with an optional port. This is discarded if the server cannot resolve it. This option is not honored by AppleShare clients <= 3.8.3. This option is disabled by default. Use with caution as this will involve a second name resolution step

-ipaddr *[ip address]* Specifies the IP address that the server should advertise (the default is the first IP address of the system). This option also allows to use one machine to advertise the AFP-over-TCP/IP settings of another machine via NBP.

-port *[port number]* Allows a different TCP port to be used for AFP-over-TCP. The default is 548.

-proxy Runs an AppleTalk proxy server for the specified AFP-over-TCP server. If the address and port aren't given, then the first IP address of the system and port 548 will be used. If you don't want the proxy server to act as a DDP server as well, set `-uamlist ""`.

-server_quantum *[number]* This specifoes the DSI server quantum. The minimum value is 303840 (0x4A2E0). The maximum value is 0xFFFFFFFFF. If you specify a value that is out of range, the default value will be set (which is the minimum). Do not change this value unless you're absolutely sure, what you're doing

-noslp Do not register this server using the Service Location Protocol (if SLP support was compiled in). This is useful if you are running multiple servers and want one to be hidden, perhaps because it is advertised elsewhere, ie. by a SLP Directory Agent.

Miscellaneous Options

-admingroup *[group]* Allows users of a certain group to be seen as the superuser when they log in. This option is disabled by default.

-authprintdir *[path]* Specifies the path to be used (per server) to store the files required to do CAP-style print authentication which papd will examine to determine if a print job should be allowed. These files are created at login and if they are to be properly removed, this directory probably needs to be umode 1777.

NOTE

`-authprintdir` will only work for clients connecting via DDP. Almost all modern Clients will use TCP.

-client_polling With this switch enabled, afpd won't advertise that it is capable of server notifications, so that connected clients poll the server every 10 seconds to detect changes in opened server windows. *Note*: Depending on the number of simultaneously connected clients and the network's speed, this can lead to a significant higher load on your network!

NOTE

Do not use this option any longer as Netatalk 2.0 correctly supports server notifications, allowing connected clients to update folder listings in case another client changed the contents

-cnidserver *[ipaddress:port]* Specifies the IP address and port of a cnid_metad server, required for CNID dbd backend. Defaults to localhost:4700.

-guestname *[name]* Specifies the user that guests should use (default is "nobody"). The name should be quoted.

-icon Use the platform-specific icon

-loginmesg *[message]* Sets a message to be displayed when clients logon to the server. The message should be in `unixcodepage` and should be quoted. Extended characters are allowed.

-nodebug Disables debugging

-sleep *[number]* AFP 3.x waits *number* hours before disconnecting clients in sleep mode. Default is 10 hours

-signature { user:<text> | **host }** Specify a server signature. This option is useful while running multiple independent instances of afpd on one machine (eg. in clustered environments, to provide fault isolation etc.). "host" signature type allows afpd generating signature automatically (based on machine primary IP address). "user" signature type allows administrator to set up a signature string manually. The maximum length is 16 characters

Example 5.6.1: Three server definitions using 2 different server signatures

```
first -signature user:USERS
second -signature user:USERS
third -signature user:ADMINS
```

First two servers will appear as one logical AFP service to the clients - if user logs in to first one and then connects to second one, session will be automatically redirected to the first one. But if client connects to first and then to third, will be asked for password twice and will see resources of both servers. Traditional method of signature generation causes two independent afpd instances to have the same signature and thus cause clients to be redirected automatically to server (s)he logged in first.

Logging Options

-[un]setuplog "*<logtype> <loglevel> [<filename>]*" Specify that the given loglevel should be applied to log messages of the given logtype and that these messages should be logged to the given file. If the filename is ommited the loglevel applies to messages passed to syslog. Each logtype may have a loglevel applied to syslog and a loglevel applied to a single file. Latter `-setuplog` settings will override earlier ones of the same logtype (file or syslog).

logtypes: Default, Core, Logger, CNID, AFP

Daemon loglevels: LOG_SEVERE, LOG_ERROR, LOG_WARN, LOG_NOTE, LOG_INFO, LOG_DEBUG, LOG_DEBUG6, LOG_DEBUG7, LOG_DEBUG8, LOG_DEBUG9, LOG_MAXDEBUG

Example 5.6.2: Some ways to change afpd's logging behaviour via -[un]setuplog

Example:

```
-setuplog "logger log_maxdebug /var/log/netatalk-logger.log"
-setuplog "afpdaemon log_maxdebug /var/log/netatalk-afp.log"
-unsetuplog "default level file"
-setuplog "default log_maxdebug"
```

Debug Options

These options are useful for debugging only.

-tickleval *[number]* Sets the tickle timeout interval (in seconds). Defaults to 30.

-timeout *[number]* Specify the number of tickles to send before timing out a connection. The default is 4, therefore a connection will timeout after 2 minutes.

Examples

Example 5.6.3: afpd.conf default configuration

```
- -transall -uamlist uams_clrtxt.so,uams_dhx.so,uams_guest.so
```

Example 5.6.4: afpd.conf MacCyrillic setup / UTF8 unix locale

```
- -transall -maccodepage mac_cyrillic -unixcodepage utf8
```

Example 5.6.5: afpd.conf setup for Kerberos V auth

```
- -transall -uamlist uams_clrtxt.so,uams_dhx.so,uams_guest.so,uams_gss.so \
-k5service afpserver -k5keytab /path/to/afpserver.keytab \
-k5realm YOUR.REALM -fqdn your.fqdn.namel:548
```

Example 5.6.6: afpd.conf letting afpd appear as three servers on the net

```
"Guest Server" -uamlist uams_guest.so -loginmesg "Welcome guest!"
"User Server" -uamlist uams_dhx.so -port 12000
"special" -notcp -defaultvol <path> -systemvol <path>
```

See also

afpd(8), afppasswd(1), AppleVolumes.default(5)

5.7 afppasswd

Name

afppasswd — netatalk password maintenance utility

Synopsis

`afppasswd [-acfn] [ -p` ***passwd file*** `] [ -u` ***minimum uid*** `]`

DESCRIPTION

afppasswd allows the maintenance of afppasswd files created by netatalk for use by some User Authentication Modules (UAMs)

afppasswd can either be called by root with parameters, or can be called by local system users with no parameters to change their passwords.

EXAMPLE

Local user changing their own password:

```
example% afppasswd
Enter NEW AFP password: (hidden)
Enter NEW AFP password again: (hidden)
afppasswd: updated password.
```

OPTIONS

-a Add a new user to the **afppasswd** file.

-c Create and/or initialize **afppasswd** file or specific user.

-f Force the current action.

-p*path* Path to **afppasswd** file.

-n If cracklib support is built into *netatalk* this option will cause cracklib checking to be disabled, if the superuser does not want to have the password run against the cracklib dictionary.

-u*minimum uid* This is the minimum *user id* (uid) that **afppasswd** will use when creating users.

SEE ALSO

afpd(8), atalkd(8).

5.8 AppleVolumes.default

Name

AppleVolumes.default — Configuration file used by **afpd**(8) to determine the shares made available through Appletalk

Description

`/etc/netatalk/AppleVolumes.default` is the configuration file used by **afpd** to determine what portions of the file system will be shared via Apple Filing Protocol, as well as their behaviour. Any line not prefixed with # is interpreted. The configuration lines are composed like:

`path` *`[ volume name ] [ options ]`*

The path name must be a fully qualified path name, or a path name using either the ~ shell shorthand or any of the substitution variables, which are listed below.

The volume name is the name that appears in the Chooser ot the "connect to server" dialog on Macintoshes to represent the appropriate share. If there are spaces in the name, it should be in quotes (i.e. "File Share"). The volume name may not exceed 27 characters in length, and cannot contain the ':' character.

NOTE

Each volume has to be configured on a **single** line.

The possible options and their meanings are:

adouble:*[v1|v2|osx]* specify the format of the metadata files, which are used for saving Mac resource fork as well. Earlier versions used AppleDouble V1, the new default format is V2. Starting with Netatalk 2.0, the scheme MacOS X uses currently (10.3.x), is also supported

NOTE

Using `adouble:osx` is **not** recommended for production use. Its only aim is to temporarely share eg. FAT32 formatted FireWire harddrives written on a Macintosh with afpd. Apple's metadata scheme lacks several essential features, so using it on the server's side will break both CNIDs and MacOS 9 compatibility

allow:*[users/groups]* The allow option allows the users and groups that access a share to be specified. Users and groups are specified, delimited by commas. Groups are designated by a @ prefix. Example: allow:user1,user2,@group

deny:*[users/groups]* The deny option specifies users and groups who are not allowed access to the share. It follows the same format as the allow option.

cnidscheme:*[backend]* set the CNID backend to be used for the volume, default is [cdb] available schemes: [cdb,dbd,last]

dbpath:*[path]* Sets the database information to be stored in path. You have to specifiy a writable location, even if the volume is read only.

maccharset:*[charset]* specifies the mac client codepage for this Volume, e.g. "MAC_ROMAN", "MAC_CYRILLIC". If not specified the setting from `afpd.conf` is inherited. This setting is only required if you need volumes, where the mac codepage differs from the one globally set in `afpd.conf`.

options:*[option]* This allows multiple options to be specified in a comma delimited format. The available options are:

limitsize Limit disk size reporting to 2GB. This can be used for older Macintoshes using newer Appleshare clients.

ro Specifies the share as being read only for all users. The .AppleDB directory has to be writeable, you can use the `-dbpath` option to relocate it.

usedots Don't do :hex translation for dot files. note: when this option gets set, certain file names become illegal. These are .Parent and anything that starts with .Apple. Also, dot files created on the unix side are marked invisible.

root_preexec_close a non-zero return code from root_preexec closes the volume immediately, preventing clients to mount/see the volume in question

preexec_close a non-zero return code from preexec close the volume being immediately, preventing clients to mount/see the volume in question

password:*[password]* This option allows you to set a volume password, which can be a maximum of 8 characters long (using ASCII strongly recommended at the time of this writing)

preexec:*[command]* command to be run when the volume is mounted, ignored for user defined volumes

postexec:*[command]* command to be run when the volume is closed, ignored for user defined volumes

root_preexec:*[command]* command to be run as root when the volume is mounted, ignored for user defined volumes

root_postexec:*[command]* command to be run as root when the volume is closed, ignored for user defined volumes

rolist:[users/groups] Allows certain users and groups to have read-only access to a share. This follows the allow option format.

rwlist:*[users/groups]* Allows certain users and groups to have read/write access to a share. This follows the allow option format.

veto:*[vetoed name]* hide files and directories,where the path matches one of the '/' delimited vetoed names. Matches are partial, e.g. path is `/abc/def/file` and veto:/abc/ will hide the file.

volcharset:*[charset]* specifies the volume codepage, e.g. "UTF8", "UTF8-MAC", "ISO-8859-15". Defaults to "UTF8".

Variable substitutions

You can use variables in both volume path and volume name.

1. if you specify an unknown variable, it will not get converted.
2. if you specify a known variable, but that variable doesn't have a value, it will get ignored.

The variables which can be used for substitutions are:

$b basename

$c client's ip or appletalk address

$d volume pathname on server

$f full name (contents of the gecos field in the passwd file)

$g group name

$h hostname

$i client's ip, without port

$s server name (this can be the hostname)

$u user name (if guest, it is the user that guest is running as)

$v volume name (either ADEID_NAME or basename of path)

$z appletalk zone (may not exist)

**$$** prints dollar sign ($)

When using variable substitution in the volume name, always keep in mind, not to exceed the 27 characters limit

Example 5.8.1: Using variable substitution when defining volumes

```
/home/groups/$g "Groupdir for $g"
~ "$f is the best one"
```

We define "groupdirs" for each primary group and use a personalized server name for homedir shares.

CNID backends

The AFP protocol mostly refers to files and directories by ID and not by name. Netatalk needs a way to store these ID's in a persistent way, to achieve this several different CNID backends are available. The CNID Databases are by default located in the `.AppleDB` folder in the volume root.

cdb "Concurrent database", backend is based on sleepycat's Berkely DB. With this backend several **afpd** deamons access the CNID database directly. Berkeley DB locking is used to synchronize access, if more than one **afpd** process is active for a volume. The drawback is, that the crash of a single **afpd** process might corrupt the database.

dbd Access to the CNID database is restricted to the **cnid_metad** daemon process. **afpd** processes communicate with the daemon for database reads and updates. If built with Berkeley DB transactions the probability for database corruption is practically zero, but performance can be slower than with `cdb`

last This backend is an exception, in terms of ID persistency. ID's are only valid for the current session. This is basically what **afpd** did in the 1.5 (and 1.6) versions. This backend is still available, as it is useful for e.g. sharing cdroms.

Warning: It is *NOT* recommended to use this backend for volumes anymore, as **afpd** now relies heavily on a persistent ID database. Aliases will likely not work and filename mangling is not supported.

Even though **./configure –help** might show that there are other CNID backends available, be warned those are likely broken or mainly used for testing. Don't use them unless you know what you're doing, they may be removed without further notice from future versions.

Charset options

With OS X Apple introduced the AFP3 protocol. One of the most important changes was that AFP3 uses unicode names encoded as UTF-8 decomposed. Previous AFP/OS versions used codepages, like MacRoman, MacCentralEurope, etc.

afpd needs a way to preserve extended macintosh characters, or characters illegal in unix filenames, when saving files on a unix filesystem. Earlier versions used the the so called CAP encoding. An extended character (>0x7F) would be converted to a :xx sequence, e.g. the Apple Logo (MacRoman: 0XF0) was saved as `:f0`. Some special characters will be converted as to :xx notation as well. '/' will be encoded to `:2f`, if `-usedots` is not specified, a leading dot '.' will be encoded as `:2e`.

This version now uses UTF-8 as the default encoding for names. Special characters, like '/' and a leading '.' will still be CAP style encoded .

The `-volcharset` option will allow you to select another volume encoding. E.g. for western users another useful setting could be -volcharset ISO-8859-15. **apfd** will accept any **iconv**(1) provided charset. If a character cannot be converted from the mac codepage to the selected volcharset, afpd will save it as a CAP encoded character. For AFP3 clients, **afpd** will convert the UTF-8 character to `-maccharset` first. If this conversion fails, you'll receive a -50 error on the mac.

Note: Whenever you can, please stick with the default UTF-8 volume format.

Compatibility with earlier versions

To use a volume created with an earlier **afpd** version, you'll have to specify the following options:

Example 5.8.2: use a 1.x style volume

```
/path/to/volume "Volname" adouble:v1 volcharset:ASCII
```

In case you used an NLS you could try using a compatible iconv charset for `-volcharset`.

Example 5.8.3: use a 1.x style volume, created with maccode.iso8859-1

```
/path/to/volume "Volname" adouble:v1 volcharset:ISO-8859-1
```

You should consider converting old style volumes to the new UTF-8/AD2 format. The safest way to do this, is to create a new volume with the default options and copy the files between this volumes with a mac.

Note: Using above example options will allow you to downgrade to 1.x netatalk again.

Note: Some 1.x NLS files used non standard mappings, e.g. `maccode.iso8859-1.adapted`. This is not supported anymore. You'll have to copy the contents of those volumes files to a Mac and then back to the netatalk server, preferably to an UTF-8 volume.

Advanced Options

The following options should only be used after serious consideration. Be sure you fully understood the, sometimes complex, consequences, before using them.

casefold:`[option]` The casefold option handles, if the case of filenames should be changed. The available options are:

`tolower` - Lowercases names in both directions.

`toupper` - Uppercases names in both directions.

`xlatelower` - Client sees lowercase, server sees uppercase.

`xlateupper` - Client sees uppercase, server sees lowercase.

options:`[option]` This allows multiple options to be specified in a comma delimited format. The available options are:

crlf Enables crlf translation for TEXT files, automatically converting macintosh line breaks into Unix ones. Use of this option might be dangerous since some older programs store binary data files as type "TEXT" when saving and switch the filetype in a second step. **Afpd** will potentially destroy such files when "erroneously" changing bytes in order to do line break translation

dropbox Allows a volume to be declared as being a "dropbox." Note that netatalk must be compiled with dropkludge support for this to function. *Warning*: This option is deprecated and might not work as expected.

mswindows Forces filename restrictions imposed by MS WinXX. *Warning*: This is *NOT* recommened for volumes mainly used by Macs. Please make sure you fully understand this option before using it.

noadouble Forces **afpd** to not create .AppleDouble directories unless macintosh metadata needs to be written. This option is only useful if you want to share files mostly used NOT by macs, causing **afpd** to not automatically create .AppleDouble subdirs containing AD header files in every directory it enters (which will it do by default).

In case, you save or change files from mac clients, AD metadata files have to be written even in case you set this option. So you can't avoid the creation of .AppleDouble directories and its contents when you give macs write access to a share and they make use of it.

Try to avoid `noadouble` whenever possible

nodev always use 0 for device number, helps when the device number is not constant across a reboot, cluster, ...

nofileid don't advertise createfileid, resolveid, deleteid calls

nohex Disables :hex translations for anything except dot files. This option makes the '/' character illegal.

prodos Provides compatibility with Apple II clients.

nostat don't stat volume path when enumerating volumes list, useful for automounting or volumes created by a preexec script.

upriv use AFP3 unix privileges. Become familiar with the new "unix privileges" AFP permissions concepts in MacOS X before using this option.

See Also

afpd.conf(5), afpd(8)

5.9 apple_cp

Name

apple_cp — Do an apple copy, copying the resource fork as well

Synopsis

/usr/bin/apple_cp SOURCE DEST /usr/bin/apple_cp SOURCE... DIRECTORY

DESCRIPTION

apple_cp is a perl script to copy SOURCE to DEST or multiple SOURCE(s) to DIRECTORY. It also copies the resource forks to the .AppleDouble directory for DEST or DIRECTORY. If the .AppleDouble directory doesn't exist for DEST or DIRECTORY it will create it.

EXAMPLES

/usr/bin/apple_cp test.text /target/directory/

/usr/bin/apple_cp test.text /target/directory/test2.text

/usr/bin/apple_cp test.text testing.text /target/directory/

REPORTING BUGS

Report bugs to the Netatalk-devel list <netatalk-devel@lists.sourceforge.net>.

SEE ALSO

apple_mv(1), apple_rm(1).

5.10 apple_mv

Name

apple_mv — Do an apple move, moving the resource fork as well

Synopsis

/usr/bin/apple_mv SOURCE DEST /usr/bin/apple_mv SOURCE... DIRECTORY

DESCRIPTION

apple_mv is a perl script to move SOURCE to DEST or multiple SOURCE(s) to DIRECTORY. It also moves the resource forks to the .AppleDouble directory for DEST or DIRECTORY. If the .AppleDouble directory doesn't exist for DEST or DIRECTORY it will create it.

EXAMPLES

/usr/bin/apple_mv test.text /target/directory/

/usr/bin/apple_mv test.text /target/directory/test2.text

/usr/bin/apple_mv test.text testing.text /target/directory/

REPORTING BUGS

Report bugs to the Netatalk-devel list <netatalk-devel@lists.sourceforge.net>.

SEE ALSO

apple_cp(1), apple_rm(1).

5.11 apple_rm

Name

apple_rm — Do an apple remove, remove the resource fork as well

Synopsis

/usr/bin/apple_rm FILE...

DESCRIPTION

apple_rm is a perl script that removes FILE(s) as well as the .AppleDouble resource fork file(s) that corresponds to FILE(s). *apple_rm* does not delete directories.

EXAMPLES

/usr/bin/apple_rm test.text

/usr/bin/apple_rm test.text testing.text

REPORTING BUGS

Report bugs to the Netatalk-devel list <netatalk-devel@lists.sourceforge.net>.

SEE ALSO

apple_cp(1), apple_mv(1).

5.12 atalk

Name

atalk — AppleTalk protocol family

Synopsis

#include <sys/types.h> #include <netatalk/at.h>

DESCRIPTION

The AppleTalk protocol family is a collection of protocols layered above the Datagram Delivery Protocol (DDP), and using AppleTalk address format. The AppleTalk family may provide SOCK_STREAM (ADSP), SOCK_DGRAM (DDP), SOCK_RDM (ATP), and SOCK_SEQPACKET (ASP). Currently, only DDP is implemented in the kernel; ATP and ASP are implemented in user level libraries; and ADSP is planned.

ADDRESSING

AppleTalk addresses are three byte quantities, stored in network byte order. The include file <*netatalk/at.h*> defines the AppleTalk address format.

Sockets in the AppleTalk protocol family use the following address structure:

struct

```
struct sockaddr_at {
short sat_family;
u_char sat_port;
struct at_addr sat_addr;
char sat_zero[ 8 ];
};
```

The port of a socket may be set with bind(2). The node for *bind* must always be *ATADDR_ANYNODE*: "this node." The net may be *ATADDR_ANYNET* or *ATADDR_LATENET*. *ATADDR_ANYNET* coresponds to the machine's "primary" address (the first configured). *ATADDR_LATENET* causes the address in outgoing packets to be determined when a packet is sent, i.e. determined late. *ATADDR_LATENET* is equivalent to opening one socket for each network interface. The port of a socket and either the primary address or *ATADDR_LATENET* are returned with getsockname(2).

SEE ALSO

bind(2), getsockname(2), atalkd(8).

5.13 atalkd

Name

atalkd — AppleTalk RTMP, NBP, ZIP, and AEP manager

Synopsis

`atalkd [-f` ***`configfile`*** `] [-1] [-2]`

Description

atalkd is responsible for all user level AppleTalk network management. This includes routing, name registration and lookup, zone lookup, and the AppleTalk Echo Protocol (similar to ping(8)). **atalkd** is typically started at boot time, out of `/etc/rc`. It first reads from its configuration file, `/etc/netatalk/atalkd.conf`. If there is no configuration file, **atalkd** will attempt to configure all available interfaces and will create a configuration file. The file consists of a series of interfaces, one per line. Lines with '#' in the first column are ignored, as are blank lines. The syntax is

interface [`-seed`] [`-phase` ***number***] [`-net` ***net-range***] [`-addr` ***address***] [`-zone` ***zonename***] ...

Note that all fields except the interface are optional. The loopback interface is configured automatically. If `-seed` is specified, all other fields must be present. Also, **atalkd** will exit during bootstrapping, if a router disagrees with its seed information. If `-seed` is not given, all other information may be overriden during autoconfiguration. If no `-phase` option is given, the default phase as given on the command line is used (the default is 2). If `-addr` is given and `-net` is not, a net-range of one is assumed.

The first -zone directive for each interface is the "default" zone. Under Phase 1, there is only one zone. Under Phase 2, all routers on the network are configured with the default zone and must agree. **atalkd** maps "*" to the default zone of the first interface. Note: The default zone for a machine is determined by the configuration of the local routers; to appear in a non-default zone, each service, e.g. **afpd**, must individually specify the desired zone. See also nbp_name(3).

Routing

If you are connecting a netatalk router to an existing AppleTalk internet, you should first contact your local network administrators to obtain appropriate network addresses.

atalkd can provide routing between interfaces by configuring multiple interfaces. Each interface must be assigned a unique net-range between 1 and 65279 (0 and 65535 are illegal, and addresses between 65280 and 65534 are reserved for startup). It is best to choose the smallest useful net-range, i.e. if you have three machines on an Ethernet, don't chose a net-range of 1000-2000. Each net-range may have an arbitrary list of zones associated with it.

Examples

Below is an example configuration file for a sun4/40. The machine has two interfaces, "le0" and "le1". The "le0" interface is configured automatically from other routers on the network. The machine is the only router for the "le1" interface.

```
le0
le1 -seed -net 9461-9471 -zone netatalk -zone Argus
```

atalkd automatically acts as a router if there is more than one interface.

Files

`/etc/netatalk/atalkd.conf` configuration file

Bugs

On some systems, **atalkd** can not be restarted.

5.14 atalkd.conf

Name

atalkd.conf — Configuration file used by atalkd(8) to determine the interfaces used by the master Netatalk daemon

DESCRIPTION

/etc/netatalk/atalkd.conf is the configuration file used by atalkd to configure the Appletalk interfaces and their behavior

Any line not prefixed with # is interpreted. The configuration lines are composed like:

Interface [options]

The simplest case is to have either no atalkd.conf, or to have one that has no active lines. In this case, atalkd should auto-discover the local interfaces on the machine. Please note that you cannot split lines.

The interface is the network interface that this to work over, such as *eth0* for Linux, or *le0* for Sun.

The possible options and their meanings are:

`-addr net.node` Allows specification of the net and node numbers for this interface, specified in Appletalk numbering format (example: `-addr 66.6`).

`-dontroute` Disables Appletalk routing. It is the opposite of `-router`.

`-net first[-last]` Allows the available net to be set, optionally as a range.

`-noallmulti` (linux only) On linux the interfaces, atalkd uses, are set to ALLMULTI by default caused by countless NICs having problems without being forced into this mode (some even don't work with all-multi set). In case, you've a NIC known to support multicasts properly, you might want to set this option causing less packets to be processed

`-phase ( 1 | 2 )` Specifies the Appletalk phase that this interface is to use (either Phase 1 or Phase 2).

`-router` Like `-seed`, but allows single interface routing. It is the opposite of `-dontroute`.

`-seed` The seed option only works if you have multiple interfaces. It also causes all missing arguments to be automagically configured from the network.

`-zone zonename` Specifies a specific zone that this interface should appear on (example: `-zone "Parking Lot"`). Please note that zones with spaces and other special characters should be enclosed in parentheses.

SEE ALSO

atalkd(8)

5.15 cnid_dbd

Name

cnid_dbd — implement access to CNID databases through a dedicated daemon process

Synopsis

`cnid_dbd` ***`dbdir ctrlfd clntfd`***

DESCRIPTION

cnid_dbd provides an interface for storage and retrieval of catalog node IDs (CNIDs) and related information to the *afpd* daemon. CNIDs are a component of Macintosh based file systems with semantics that map not easily onto Unix file systems. This makes separate storage in a database necessary. **cnid_dbd** is part of the *CNID backend* framework of *afpd* and implements the *dbd* backend.

cnid_dbd is never started via the command line or system startup scripts but only by the *cnid_metad* daemon. There is at most one instance of **cnid_dbd** per netatalk volume.

cnid_dbd uses the *Berkleley DB* database library and optionally supports transactionally protected updates if the netatalk package is compiled with the appropriate options. Using the *dbd* backend without transactions will protect the CNID database against unexpected crashes of the *afpd* daemon. Using the *dbd* backend with transactions will avoid corruption of the CNID database even if the system crashes unexpectedly.

cnid_dbd uses the same on-disk database format as the *cdb* backend. It is therefore possible to switch between the two backends as necessary.

cnid_dbd inherits the effective userid and groupid from *cnid_metad* on startup, which is normally caused by *afpd* serving a netatalk volume to a client. It changes to the *Berkleley DB* database home directory *dbdir* that is associated with the volume. If the userid inherited from *cnid_metad* is 0 (root), **cnid_dbd** will change userid and groupid to the owner and group of the database home directory. Otherwise, it will continue to use the inherited values. **cnid_dbd** will then attempt to open the database and start serving requests using filedescriptor *clntfd*. Subsequent instances of *afpd* that want to access the same volume are redirected to the running **cnid_dbd** process by *cnid_metad* via the filedescriptor *ctrlfd*.

cnid_dbd can be configured to run forever or to exit after a period of inactivity. If **cnid_dbd** receives a TERM or an INT signal it will exit cleanly after flushing dirty database buffers to disk and closing *Berkleley DB* database environments. It is safe to terminate **cnid_dbd** this way, it will be restarted when necessary. Other signals are not handled and will cause an immediate exit, possibly leaving the CNID database in an inconsistent state (no transactions) or losing recent updates during recovery (transactions).

If transactions are used the *Berkleley DB* database subsystem will create files named log.xxxxxxxxxx in the database home directory *dbdir*, where xxxxxxxxxx is a monotonically increasing integer. These files contain information to replay database changes and are not automatically removed, unless the *logfile_autoremove* option is specified in the *db_param* configuration file (see below). Please see the sections *Database and log file archival*, *Log file removal* and the documentation of the *db_archive* command line utility in the Berkeley DB Tutorial and Reference for information when and how it is safe to remove these files manually.

Do not use **cnid_dbd** for databases on NFS mounted file systems. It makes the whole point of securing database changes properly moot. Use the dbdir: Option in the appropriate *AppleVolumes* configuration file to put the database onto a local disk.

CONFIGURATION

cnid_dbd reads configuration information from the file *db_param* in the database directory *dbdir* on startup. If the file does not exist or a parameter is not listed, suitable default values are used. The format for a single parameter is the parameter name, followed by one or more spaces, followed by the parameter value, followed by a newline. The following parameters are currently recognized:

logfile_autoremove This flag is ignored unless transactional support is enabled. If set to 1, unused Berkeley DB transactional logfiles (log.xxxxxxxxxx in the database home directory) are removed on startup of **cnid_dbd**. This is usually safe if the content of the database directory is backed up on a regular basis. Default: 0.

cachesize Determines the size of the Berkeley DB cache in kilobytes. Default: 1024. Each **cnid_dbd** process grabs that much memory on top of its normal memory footprint. It can be used to tune database performance. The *db_stat* utility with the `-m` option that comes with Berkely DB can help you determine wether you need to change this value. The default is pretty conservative so that a large percentage of requests should be satisfied from the cache directly. If memory is not a bottleneck on your system you might want to leave it at that value. The *Berkeley DB Tutorial and Reference Guide* has a section *Selecting a cache size* that gives more detailed information.

nosync This flag is ignored unless transactional support is enabled. If it is set to 1, transactional changes to the database are not synchronously written to disk when the transaction completes. This will increase performance considerably at the risk of recent changes getting lost in case of a crash. The database will still be consistent, though. See *Transaction Throughput* in the Berkeley DB Tutorial for more information. Default: 0.

flush_frequency, flush_interval *flush_frequency* (Default: 100) and *flush_interval* (Default: 30) control how often changes to the database are written to the underlying database files if no transactions are used or how often the transaction system is checkpointed for transactions. Both of these operations are performed if either i) more than *flush_frequency* requests have been received or ii) more than *flush_interval* seconds have elapsed since the last save/checkpoint. If you use transactions with *nosync* set to zero these parameters only influence how long recovery takes after a crash, there should never be any lost data. If *nosync* is 1, changes might be lost, but only since the last checkpoint. Be careful to check your harddisk configuration for on disk cache settings. Many IDE disks just cache writes as the default behaviour, so even flushing database files to disk will not have the desired effect.

fd_table_size is the maximum number of connections (filedescriptors) that can be open for *afpd* client processes in *cnid_dbd*. Default: 16. If this number is exceeded, one of the existing connections is closed and reused. The affected *afpd* process will transparently reconnect later, which causes slight overhead. On the other hand, setting this parameter too high could affect performance in **cnid_dbd** since all descriptors have to be checked in a `select()` system call, or worse, you might exceed the per process limit of open file descriptors on your system. It is safe to set the value to 1 on volumes where only one *afpd* client process is expected to run, e.g. home directories.

idle_timeout is the number of seconds of inactivity before an idle **cnid_dbd** exits. Default: 600. Set this to 0 to disable the timeout.

SEE ALSO

cnid_metad(8), afpd(8)

5.16 cnid_metad

Name

cnid_metad — start cnid_dbd daemons on request

Synopsis

```
cnid_metad [-d] [ -h hostname ] [ -p port ] [ -u user ] [ -g group ] [ -s
    cnid_dbdpathname ]
```

DESCRIPTION

cnid_metad waits for requests from *afpd* to start up instances of the *cnid_dbd* daemon. It keeps track of the status of a *cnid_dbd* instance once started and will restart it if necessary. **cnid_metad** is normally started at boot time from `/etc/rc` or equivalent and runs until shutdown. *afpd* needs to be configured with the `-cnidserver` option in *afpd.conf* in order to access *cnid_metad.* It is possible to run more than one instance of **cnid_metad** on the same machine if different values for the interface and/or port are specified with the `-h` and `-p` options.

OPTIONS

-d *cnid_metad will remain in the foreground and* will also leave the standard input, standard output and standard error file descriptors open. Useful for debugging.

-h*hostname* Use *hostname* as the network interface for requests as opposed to the default *localhost*

-p*port* Use *port* as the port number for reqests. Default is 4700.

-u*user* Switch to the userid of *user* before serving requests. This userid will be inherited by all *cnid_dbd* daemon processes started.

-u*group* Switch to the groupid of *group* before serving requests. This groupid will be inherited by all *cnid_dbd* daemon processes started. Both *user* and *group* must be specified as strings.

-s*cnid_dbd pathname* Use *cnid_dbd pathname* as the pathname of the executeable of the *cnid_dbd* daemon. The default is */usr/sbin/cnid_dbd.*

CAVEATS

The number of *cnid_dbd* subprocecesses is currently limited to 128. This restriction will be lifted in the future.

cnid_metad does not block or catch any signals apart from SIGPIPE. It will therefore exit on most signals received. This will also cause all instances of *cnid_dbd's* started by that **cnid_metad** to exit gracefully. Since state about and IPC access to the subprocesses is only maintained in memory by **cnid_metad** this is desired behaviour. As soon as **cnid_metad** is restarted *afpd* processes will transparently reconnect.

SEE ALSO

cnid_dbd(8), afpd(8)

5.17 getzones

Name

getzones — list AppleTalk zone names

Synopsis

`getzones [-m | -l] [address]`

DESCRIPTION

Getzones is used to obtain a list of AppleTalk zone names using the Zone Information Protocol (ZIP). It sends a GetZoneList request to an AppleTalk router. By default, it sends the request to the locally running atalkd(8).

OPTIONS

-m List the name of the local zone only; this is accomplished by sending a ZIP GetMyZone request.

-l List the local zones; this is accomplished by sending a GetLocalZones request.

address Contact the AppleTalk router at *address*. *address* is parsed by atalk_aton(3).

SEE ALSO

atalk_aton(3), zip(4), atalkd(8).

5.18 megatron

Name

megatron, unhex, unbin, unsingle, hqx2bin, single2bin, macbinary — Macintosh file format transformer

Synopsis

`megatron [sourcefile...] unbin [sourcefile...] unhex [sourcefile...] unsingle [sourcefile...] hqx2bin [sourcefile...] single2bin [sourcefile...] macbinary [sourcefile...]`

DESCRIPTION

megatron is used to transform files from BinHex, MacBinary, AppleSingle, or **netatalk** style AppleDouble formats into MacBinary or *netatalk* style AppleDouble formats. The *netatalk* style AppleDouble format is the file format used by *afpd,* the *netatalk* Apple Filing Protocol (AppleShare) server. BinHex, MacBinary, and AppleSingle are commonly used formats for transferring Macintosh files between machines via email or file transfer protocols. **megatron** uses its name to determine what type of tranformation is being asked of it.

If **megatron** is called as **unhex** , **unbin** or **unsingle**, it tries to convert file(s) from BinHex, MacBinary, or AppleSingle into AppleDouble format. BinHex is the format most often used to send Macintosh files by e-mail. Usually these files have an extension of ".hqx". MacBinary is the format most often used by terminal emulators "on the fly" when transferring Macintosh files in binary mode. MacBinary files often have an extension of ".bin". Some Macintosh LAN-based email packages use uuencoded AppleSingle format to "attach" or "enclose" files in email. AppleSingle files don't have a standard filename extension.

If **megatron** is called as **hqx2bin**, **single2bin**, or **macbinary**, it will try to convert the file(s) from BinHex, AppleSingle, or AppleDouble into MacBinary. This last translation may be useful in moving Macintosh files from your **afpd** server to some other machine when you can't copy them from the server using a Macintosh for some reason.

If **megatron** is called with any other name, it uses the default translation, namely **unhex**.

If no source file is given, or if *sourcefile* is '**-**', and if the conversion is from a BinHex or MacBinary file, **megatron** will read from standard input.

The filename used to store any output file is the filename that is encoded in the source file. MacBinary files are created with a ".bin" extension. In the case of conflicts, the old file is overwritten!

SEE ALSO

afpd(8)

5.19 nbp

Name

nbplkup, nbprgstr, nbpunrgstr — access NBP database

Synopsis

`nbplkup [-r maxresponses] [-A address] [-m maccodepage] nbpname`

`nbprgstr [-A address] [-p port] [-m maccodepage] nbpname`

`nbpunrgstr [-A address] [-m maccodepage] nbpname`

Description

nbprgstr registers *nbpname* with atalkd(8), at the given *port*. **nbpunrgstr** informs *atalkd* that *nbpname* is no longer to be advertised.

nbplkup displays up to `maxresponses` (default 1000) entities registered on the AppleTalk internet. `nbpname` is parsed by nbp_name(3). An '=' for the *object* or *type* matches anything, and an '*' for *zone* means the local zone. The default values are taken from the *NBPLKUP* environment variable, parsed as an `nbpname`.

Environment Variables

NBPLKUP default nbpname for nbplkup

ATALK_MAC_CHARSET the codepage used by the clients on the Appletalk network

ATALK_UNIX_CHARSET the codepage used to display extended characters on this shell.

Example

Find all devices of type *LaserWriter* in the local zone.

```
example% nbplkup :LaserWriter
               Petoskey:LaserWriter        7942.129:218
             Gloucester:LaserWriter        8200.188:186
                 Rahway:LaserWriter        7942.2:138
             517 Center:LaserWriter        7942.2:132
                  ionia:LaserWriter        7942.2:136
     Evil DEC from Hell:LaserWriter        7942.2:130
              Hamtramck:LaserWriter        7942.2:134
         Iron Mountain :LaserWriter        7942.128:250
example%
```

See also

nbp_name(3), nbp(4), zip(4), atalkd(8).

5.20 netatalk.conf

Name

netatalk.conf — Configuration file used by netatalk(8) to determine its general configuration

DESCRIPTION

/etc/netatalk/netatalk.conf is the configuration file used by afpd to determine what portions of the file system will be shared via Appletalk, as well as their behaviors.

Any line not prefixed with # is interpreted. The configuration lines are composed like:

option = value

The possible options and their meanings are:

AFPD_GUEST Sets the id of the guest user to a local user on the system.

AFPD_MAX_CLIENTS Sets the maximum number of clients that can simultaneously connect to the server.

AFPD_RUN Enables the afpd daemon if set to "yes". This should be enabled if you are planning on using netatalk as a file server.

AFPD_UAM_LIST Sets the default UAMs for afpd (and papd, if printer authentication is compiled in) to use.

Example: AFPD_UAMLIST="-U uams_guest.so,uams_randnum.so"

CNID_METAD_RUN Enables the cnid_metad daemon if set to "yes". This should be enabled if you are going to use the dbd CNID backend.

ATALK_BGROUND "yes" will set netatalk to initialize in the background, and "no" will cause normal initialization.

ATALK_NAME Sets the machines' Appletalk name.

ATALK_ZONE Sets the machines' Appletalk zone.

ATALKD_RUN Enables the atalkd daemon if set to "yes". This should be enabled if you are planning on providing Appletalk services.

PAPD_RUN Enables the papd daemon if set to "yes". This should be enabled if you are planning on using netatalk as a print server.

ATALK_MAC_CHARSET Set the Mac client codepage, used by atalkd and papd to convert extended characters from the Unix to the Mac codepage.

ATALK_UNIX_CHARSET Set the Unix codepage, used by atalkd and papd to convert extended characters from the Unix to the Mac codepage. Has to match the codepage of the configuration files.

SEE ALSO

atalkd(8), atalkd.conf(5)

5.21 netatalk-config

Name

netatalk-config — script to get information about the installed version of netatalk

Synopsis

```
netatalk-config [ -prefix [=DIR]] [ -exec_prefix [=DIR]] [-help] [-version] [-libs]
     [-libs-dirs] [-libs-names] [-cflags] [-macros]
```

DESCRIPTION

netatalk-config is a tool that is used to configure to determine the compiler and linker flags that should be used to compile and link programs that use the *netatalk* run-time libraries.

OPTIONS

netatalk-config accepts the following options:

-help Print a short help for this command and exit.

-version Print the currently installed version of *netatalk* on the standard output.

-libs Print the linker flags that are necessary to link against the *netatalk* run-time libraries.

-libs-dirs Print only the -l/-R part of –libs.

-libs-names Print only the -l part of –libs.

-cflags Print the compiler flags that are necessary to compile a program linked against the *netatalk* run-time libraries.

-macros Print the *netatalk* m4 directory.

-prefix=PREFIX If specified, use PREFIX instead of the installation prefix that *netatalk* was built with when computing the output for the –cflags and –libs options. This option is also used for the exec prefix if –exec-prefix was not specified. This option must be specified before any –libs or –cflags options.

-exec_prefix=PREFIX If specified, use PREFIX instead of the installation exec prefix that *netatalk* was built with when computing the output for the –cflags and –libs options. This option must be specified before any –libs or –cflags options.

COPYRIGHT

Man page adapted for **netatalk-config** by Sebastian Rittau in 2001.

5.22 pap

Name

pap, papstatus — client interface to remote printers using Printer Access Protocol

Synopsis

```
pap [ -A address ] [-c] [-d] [-e] [-E] [ -p nbpname ] [ -s statusfile ] [-w] [-W] [FILES]
    papstatus [ -p nbpname ]
```

DESCRIPTION

pap is used to connect and send files to an AppleTalk connected printer using the Apple Printer Access Protocol (PAP). When **pap** starts execution, it tries to open a session with the printer using PAP, and then downloads the *files* to the printer.

If no *files* are given on the command line, **pap** begins reading from standard input.

If no printer is specified on the command line, **pap** looks for a file called .paprc in the current working directory and reads it to obtain the *nbpname* of a printer. Blank lines and lines that begin with a '#' are ignored. *type* and *zone* default to *LaserWriter* and the zone of the local host, respectively.

Note that **pap** is designed to be useful as a communication filter for sending lpd(8) spooled print jobs to AppleTalk connected printers. See psf(8) for hints on how to use it this way.

OPTIONS

-A*address* Connect to the printer with Appletalk address *address* and do not consult the .paprc file to find a printer name. See atalk_aton(3) for the syntax of *address*.

-c Take cuts. The PAP protocol specified a simple queuing procedure, such that the clients tell the printer how long they have been waiting to print. This option causes **pap** to lie about how long it has been waiting.

-d Enable debug output.

-e Send any message from the printer to stderr instead of stdout. psf(8) invokes **pap** with this option.

-E Don't wait for EOF from the printer. This option is useful for printers which don't implement PAP correctly. In a correct implementation, the client side should wait for the printer to return EOF before closing the connection. Some clients don't wait, and hence some printers have related bugs in their implementation.

-p *nbpname* Connect to the printer named *nbpname* and do not consult the .paprc file to find a printer name. See nbp_name(3) for the syntax of *nbpname*.

-s *statusfile* Update the file called *statusfile* to contain the most recent status message from the printer. **pap** gets the status from the printer when it is waiting for the printer to process input. The *statusfile* will

contain a single line terminated with a newline. This is useful when **pap** is invoked by psf(8) within *lpd*'s spool directory.

-w Wait for the printer status to contain the word "waiting" before sending the job. This is to defeat printer-side spool available on HP IV and V printers.

-W Wait for the printer status to contain the word "idle" before sending the job. This is to defeat printer-side spool available on HP IV and V printers.

FILES

.paprc file read to obtain printer name if not specified on command line

SEE ALSO

nbp_name(3), atalk_aton(3), lpd(8), psf(8).

5.23 papd

Name

papd — AppleTalk print server daemon

Synopsis

```
papd [-d] [-f configfile] [-p printcap]
```

Description

papd is the AppleTalk printer daemon. This daemon accepts print jobs from AppleTalk clients (typically Macintosh computers) using the Printer Access Protocol (PAP). When used with System V printing systems, **papd** spools jobs directly into an lpd(8) spool directory and wakes up **lpd** after accepting a job from the network to have it re-examine the appropriate spool directory. The actual printing and spooling is handled entirely by **lpd**.

papd can also pipe the print job to an external program for processing, and this is the preferred method on systems not using CUPS to avoid compatibility problems with all the flavours of **lpd** in use.

As of version 2.0, CUPS is also supported. Simply using *cupsautoadd* as first papd.conf entry will share all CUPS printers automagically using the PPD files configured in CUPS. It ist still possible to overwrite these defaults by individually define printer shares. See papd.conf(5) for details

papd is typically started at boot time, out of system init scripts. It first reads from its configuration file, `/etc/netatalk/papd.conf`. The file is in the same format as `/etc/printcap`. See printcap(5) for details. The name of the entry is registered with NBP.

The following options are supported:

Name	Type	Default	Description
pd	str	'.ppd'	Pathname to PPD file
pr	str	'lp'	LPD or CUPS printer name (or pipe to a print command)
op	str	'operator'	Operator name for LPD spooling
au	bool	false	Whether to do authenticated printing or not
ca	str	NULL	Pathname used for CAP-style authentification
sp	bool	false	PSSP-style authetication
am	str	NULL	UAMS to use for authentication
pa	str	NULL	Printer's AppleTalk address
co	str	NULL	CUPS options as supplied to the lp(1) command with "-o"
fo	bool	false	adjust lineending for foomatic-rip

If no configuration file is given, the hostname of the machine is used as the NBP name and all options take their default value.

Options

-d Do not fork or disassociate from the terminal. Write some debugging information to stderr.

-f *`configfile`* Consult *`configfile`* instead of `/etc/netatalk/papd.conf` for the configuration information.

-p *`printcap`* Consult *`printcap`* instead of `/etc/printcap` for LPD configuration information.

Notes

PSSP (Print Server Security Protocol) is an authentication protocol carried out through postscript printer queries to the print server. Using PSSP requires LaserWriter 8.6.1 or greater on the client mac. The user will be prompted to enter their username and password before they print. It may be necessary to re-setup the printer on each client the first time PSSP is enabled, so that the client can figure out that authentication is required to print. You can enable PSSP on a per-printer basis. PSSP is the recommended method of authenticating printers as it is more robust than CAP-style authentication, described below.

CAP-style authentication gets its name from the method the CAP (Columbia APpletalk) package used to authenticate its mac clients' printing. This method requires that a user login to a file share before they print. **afpd** records the username in a temporary file named after the clients Appletalk address, and it deletes the temporary file when the user disconnects. Therefore CAP style authentification will *not* work for clients connecting over to **afpd** via TCP/IP. **papd** gets the username from the file with the same Appletalk address as the machine connecting to it. CAP-style authentication will work with any mac client. If both CAP and PSSP are enabled for a particular printer, CAP will be tried first, then it will fall back to PSSP.

The list of UAMS to use for authentication (specified with the 'am' option) applies to all printers. It is not possible to define different authentication methods on each printer. You can specify the list of UAMS multiple times, but only the last setting will be used. Currently only uams_guest.so and uams_clrtxt.so are supported as printer authentication methods. The guest method requires a valid username, but not a password. The Cleartext UAM requires both a valid username and the correct password.

NOTE

As of this writing, Mac OS X makes no use of PSSP authentication any longer. CAP-style authentication normally won't be an option, too caused by the use of AFP over TCP these days.

Files

`/etc/netatalk/papd.conf` Default configuration file.

`/etc/printcap` Printer capabilities database.

`.ppd` PostScript Printer Description file. papd answers configuration and font queries from printing clients by consulting the configured PPD file. Such files are available for download from Adobe, Inc. (http://www.adobe.com/sup <http://www.adobe.com/support/downloads/main.html#Printer>), or from the printer's manufacturer. If no PPD file is configured, papd will return the default answer, possibly causing the client to send excessively large jobs.

Caveats

papd accepts characters with the high bit set (a full 8-bits) from the clients, but some PostScript printers (including Apple Computer's LaserWriter family) only accept 7-bit characters on their serial interface by default. The same applies for some printers when they're accessed via TCP/IP methods (remote LPR or socket). You will need to configure your printer to accept a full 8 bits or take special precautions and convert the printjob's encoding (eg. by using *co="protocol=BCP"* when using CUPS 1.1.19 or above).

When printing clients run MacOS 10.2 or above, take care that PPDs do not make use of **cupsFilter:* comments unless the appropriate filters are installed at the client's side, too (remember: Starting with 10.2 Apple chose to integrate CUPS into MacOS X). For in-depth information on how CUPS uses PPDs see chapter 3.4 in http://tinyurl.com/zbxn <http://tinyurl.com/zbxn>)

See also

lpr(1),lprm(1),printcap(5),lpc(8),lpd(8), lp(1).

5.24 papd.conf

Name

papd.conf — Configuration file used by papd(8) to determine the configuration of printers used by the Netatalk printing daemon

DESCRIPTION

/etc/netatalk/papd.conf is the configuration file used by papd to configure the printing services offered by netatalk. Please note that papd must be enabled in */etc/netatalk/netatalk.conf* for this to take any effect. *papd* shares the same defaults as lpd on many systems, but not Solaris.

Any line not prefixed with # is interpreted. The configuration lines are composed like:

printername:[options]

On systems running a System V printing system the simplest case is to have either no papd.conf, or to have one that has no active lines. In this case, atalkd should auto-discover the local printers on the machine. Please note that you can split lines by using *fR*.

printername may be just a name (*Printer 1*), or it may be a full name in nbp_name format (*Printer 1:LaserWriter@My Zone*). If more than 15 printers are defined, you should explicitly define the zone for each printer. Otherwise, the Mac Chooser not show all the printers.

Systems using a BSD printing system should make use of a pipe to the printing command in question within the *pr* option (eg. *pr=|/usr/bin/lpr -J%J -u%U*). Note: When printing using a pipe, papd recognizes several wildcards: %F will be replaced by the name present in the "%%For:" comment in the PostScript stream, same with %J for the "%%Title:" comment. %U will be substituted with the login name (the latter applies only when authenticated printing is in effect)

When CUPS support is compiled in, then *cupsautoadd* as the first entry in papd.conf will automagically share all CUPS printers by papd utilizing the PPDs assigned in CUPS (customizable – see below). This can be overwritten for individual printers by subsequently adding individual entries using the CUPS queue name as *pr* entry. Note: CUPS support is mutually exclusive with System V support described above

The possible options are colon delimited (*:*), and lines must be terminated with colons. The possible options and flags are:

am=(uams list) The *am* option allows specific UAMs to be specified for a particular printer. It has no effect if the *au* flag is not present or if papd authentication was not built into netatalk. Note: possible values are *uams_guest.so* and *uams_clrtxt.so* only. The first method requires a valid username, but no password. The second requires both a valid username and the correct password

au If present, this flag enables authentication for the printer. Please note that papd authentication must be built into netatalk for this to take effect.

co=(CUPS options) The *co* option allows options to be passed through to CUPS (eg. *co="protocol=TBCP"* or *co="raw"*).

cupsautoadd[:type][@zone] If used as the first entry in papd.conf this will share all CUPS printers via papd. type/zone settings as well as other parameters assigned to this special printer share will apply to all CUPS printers. Unless the *pd* option is set, the CUPS PPDs will be used. To overwrite these global settings for individual printers simply add them subsequently to papd.conf and assign different settings

fo If present, this flag enables a hack to translate line endings originating from pre Mac OS X LaserWriter drivers to let *foomatic-rip* recognize foomatic PPD options set in the printer dialog. Attention: Use with caution since this might corrupt binary print jobs!

op=(operator) This specifies the operator name, for lpd spooling.

pa=(appletalk address) Allows specification of Appletalk addresses. Usually not needed.

pd=(path to ppd file) Specifies a particular PPD (printer description file) to associate with the selected printer.

pr=(lpd/CUPS printer name or pipe command) Sets the *lpd* or *CUPS* printer that this is spooled to.

Examples

Unless CUPS support has been compiled in (which is default from Netatalk 2.0 on) one simply defines the lpd queue in question by setting the `pr` parameter to the queue name, in the following example "ps". If no `pr` parameter is set, the default printer will be used.

Example 5.24.1: papd.conf System V printing system examples

The first spooler is known by the AppleTalk name Mac Printer Spooler, and uses a PPD file located in `/usr/share/lib/ppd`. In addition, the user mcs will be the owner of all jobs that are spooled. The second spooler is known as HP Printer and all options are the default.

```
Mac Printer Spooler:\
   :pr=ps:\
   :pd=/usr/share/lib/ppd/HPLJ_4M.PPD:\
   :op=mcs:

HP Printer:\
   :
```

An alternative to the technique outlined above is to direct papd's output via a pipe into another program. Using this mechanism almost all printing systems can be driven. Netatalk supplies three "wildcards" that get substituted with values of the already printed job: `%F`, `%U` and `%J`. Using these wildcards, one can pass those parameters directly to programs or implement small wrapper scripts to call the printing system in question.

Example 5.24.2: papd.conf examples using pipes

The first spooler is known as HP 8100. It pipes the print job to **/usr/bin/lpr** for printing using the value of the *%%Title:* comment as job name. PSSP authenticated printing is enabled, as is CAP-style authenticated printing. Both methods support guest and cleartext authentication as specified by the '`am`' option. The PPD used is `/etc/atalk/ppds/hp8100.ppd`. The second spooler is called "Dump PostScript" and uses a pipe to **cat** to send the raw PostScript code into the user's home directory into a file called like the printjob.

```
HP 8100:\
   :pr=|/usr/bin/lpr -Plp -J"%J":\
   :sp:\
   :ca=/tmp/print:\
   :am=uams_guest.so,uams_pam.so:\
   :pd=/etc/atalk/ppds/hp8100.ppd:

Dump PostScript:LaserWriter@Server:\
   :pr=|cat >/home/%U/%J-prn.out:\
   :pd=/usr/share/lib/ppd/mooralana.ppd:\
   :sp:au:op=lp:\
   :am=uams_clrtxt.so:
```

Starting with Netatalk 2.0 direct CUPS integration is available. In this case, defining only a queue name as `pr` parameter won't invoke the SysV lpd daemon but uses CUPS instead. Unless a specific PPD has been assigned using the `pd` switch, the PPD configured in CUPS will be used by **papd**, too.

There exists one special share named "cupsautoadd". If this is present as the first entry then all available CUPS queues will be served automagically using the parameters assigned to this global share. But subsequent printer definitions can be used to override these global settings for individual spoolers.

Example 5.24.3: papd.conf CUPS examples

The first entry sets up automatic sharing of all CUPS printers. All those shares appear in the zone "1st floor" and since no additional settings have been made, they use the CUPS printer name as NBP name and use the PPD configured in CUPS. The second entry defines different settings for one single CUPS printer. It's NBP name is differing from the printer's name and the registration happens in another zone.

```
cupsautoadd@1st floor:op=root:

Boss' LaserWriter@2nd floor:\
   :pr=laserwriter-chief:
```

SEE ALSO

papd(8), atalkd.conf(5), lpd(8), lpoptions(8)

5.25 papstatus

Name

papstatus — get the status of an AppleTalk-connected printer

Synopsis

`/usr/sbin/papstatus [-d] [ -p printer ] [retrytime]`

DESCRIPTION

papstatus is used to obtain the current status message from an AppleTalk connected printer. It uses the Printer Access Protocol (PAP) to obtain the status information.

If no printer is specified on the command line, *papstatus* looks for a file called .paprc in the current directory and reads it to obtain the name of a printer. The .paprc file should contain a single line of the form *object:type@zone* where each of *object*, *:type*, and *@zone* are optional. *type* and *zone* must be proceeded by *':'* and '@' respectively. Blank lines and lines the begin with a '#' are ignored. *type* and *zone* default to *LaserWriter* and the zone of the local host, respectively.

OPTIONS

-d Turns on a debugging mode that prints some extra information to standard error.

-p *printer* Get status from *printer* (do not consult any .paprc files to find a printer name). The syntax for *printer* is the same as discussed above for the .paprc file.

retrytime Normally, *papstatus* only gets the status from the printer once. If *retrytime* is specified, the status is obtained repeatedly, with a sleep of *retrytime* seconds between inquiring the printer.

FILES

.paprc file that contains printer name

SEE ALSO

nbp(1), pap(1)

5.26 psf

Name

psf — PostScript filter

Synopsis

`psf [ -n name ] [ -h host ] [ -w width ] [ -l length ] [ -i indent ] [-c]`

DESCRIPTION

psf is an *lpd* filter for PostScript printing. **psf** interprets the name it was called with to determine what filters to invoke. First, if the string "pap" appears anywhere in the name, **psf** invokes *pap* to talk to a printer via AppleTalk. Next, if the string "rev" appears, **psf** invokes *psorder* to reverse the pages of the job. Finally, if **psf** was called with a filter's name as the leading string, it invokes that filter. If there is no filter to run, **psf** examines the magic number of the input, and if the input is not PostScript, converts it to PostScript.

KLUDGE

In the default configuration, **psf** supports two kludges. The first causes **psf** to check its name for the letter 'm'. If this letter is found and accounting is turned on, **psf** calls *pap* twice, once to get an initial page count and to print the job, and another time to get a final page count. This is a work-around for bugs in a variety of PAP implementions that cause printers to never properly close the PAP output file. A notable example is any printer by Hewlett-Packard.

The second kludge causes **psf** to examine its name for the letter 'w'. If this letter is found and accounting is turned on, **psf** calls *pap* with the `-w` flag. This flag causes *pap* to wait until the printer's status contains the string 'idle'. Once this string is found, the job is printed as normal. This kludge is a work-around for printers, notably Hewlett-Packard's LaserJet IV, which will report a page count while a previous jobs is still printing.

EXAMPLE

The sample *printcap* entry below invokes **psf** to print text files, PostScript files, *troff*'s C/A/T output, and *TeX*'s DVI output, to an AppleTalk connected LaserWriter Plus. Since the LaserWriter Plus stacks pages in descending order, we reverse the pages and print the burst page last.

```
laser|lp|LaserWriter Plus on AppleTalk:\
  :sd=/usr/spool/lpd/laser:\
  :lp=/usr/spool/lpd/laser/null:\
  :lf=/var/adm/lpd-errs:pw#80:hl:\
  :of=/usr/libexec/ofpap:\
  :if=/usr/libexec/ifpaprev:\
  :tf=/usr/libexec/tfpaprev:\
  :df=/usr/libexec/dfpaprev:
```

Note that if the host in question spools to more than one AppleTalk printer, `/dev/null` should not be used for the *lp* capability. Instead, a null device should be created with *mknod* for each printer, as has been done above.

Finally, there is a file in the spool directory, `/var/spool/lpd/laser`, called .paprc, which *pap* reads for the AppleTalk name of the printer.

SEE ALSO

psorder(1), printcap(5), lpd(1), mknod(1), pap(1).

5.27 psorder

Name

psorder — PostScript pageorder filter

Synopsis

`psorder [-duf]` ***`sourcefile`***

DESCRIPTION

psorder is a filter that re-orders the pages of a PostScript document. The result is written to the standard output. By default, documents are processed into ascending order (the lowest numbered page is printed first). Some PostScript documents specify that the order of their pages should never be changed; the default action of **psorder** is to follow this specification.

If no source file is given, or if *sourcefile* is '-', **psorder** reads from the standard input file.

OPTIONS

-d Re-order the pages of the document in downward or descending order. This is typically used to change the order of a document to be printed by a printer that stacks pages face-up, such as an Apple LaserWriter or LaserWriter Plus.

-u Specifies forward order, and is the default. It is used to try and ensure the correct ordering when a document will be printed by a printer that stacks the pages face-down.

-f Force **psorder** to re-order the pages, even if the document claims that its page order is not to be trifled with. This option should only be used experimentally, as it may cause documents to print incorrectly.

SEE ALSO

psf(8), lpr(1).

5.28 timelord

Name

timelord — Macintosh time server daemon

SYNTAX

timelord [-d] [-n *filename*]

DESCRIPTION

timelord is a simple time server for Macintosh computers that use the *tardis* client.

OPTIONS

-d Debug mode, i.e. don't disassociate from controlling TTY.

-n *nbpname* Register this server as *nbpname*. This defaults to the hostname.

5.29 timeout

Name

timeout — Send a signal to a program after a certain time

SYNTAX

timeout [-s *signal*] *seconds program* [*args*]

DESCRIPTION

timeout executes a *program* (with arguments *args*) and sends a *signal* to it after a certain amount of *seconds*.

OPTIONS

-s *signal* Signal to send to the spawned process. This can be a numerical or symbolic ID. This defaults to `TERM`.

EXAMPLES

timeout 10 pap foo.ps Execute "pap foo.ps" and send a `SIGTERM` if *pap* doesn't return after 10 seconds.

timeout -s HUP 60 sh Spawn a shell and send a hangup signal after one minute.

timeout -s 9 10 evilprog Execute a program and *KILL* it if it doesn't quit after 10 seconds.

5.30 uniconv

Name

uniconv — convert Netatalk volume encoding

Synopsis

`uniconv [-ndv] -c cnidbackend -f fromcode -t tocode [-m maccode] volumepath`

Description

uniconv converts the volume encoding of *`volumepath`* from the *`fromcode`* to the *`tocode`* encoding.

Options

-c CNID backend used on this volume, usually cdb or dbd. Should match the backend selected with afpd for this volume. If not specified, the default CNID backend 'cdb' is used

-d don't CAP encode leading dots (:2e), equivalent to `usedots` in AppleVolumes.default(5)

-f encoding to convert from, use ASCII for CAP encoded volumes

-h display help

-m Macintosh client codepage, required for CAP encoded volumes. Defaults to 'MAC_ROMAN'

-n 'dry run', don't do any real changes

-t volume encoding to convert to, e.g. UTF8

-v verbose output, use twice for maximum logging.

-V print version and exit

WARNING

Setting the wrong options might render your data unusable!!! Make sure you know what you are doing. Always backup your data first.

It is ***strongly*** recommended to do a 'dry run' first and to check the output for conversion errors.

afpd(8) should *not* be running while you change the volume encoding. Remember to change `volcodepage` in AppleVolumes.default(5) to the new codepage, before restarting afpd.

USE AT YOUR OWN RISK!!!

Selectable charsets

Netatalk provides internal support for UTF-8 (pre- and decomposed) and CAP. If you want to use other charsets, they must be provided by iconv(1)

uniconv also knows iso-8859.adapted, an old style 1.x NLS widely used. This is only intended for upgrading old volumes, afpd(8) cannot handle iso-8859.adapted anymore.

CNID background

The CNID backends maintains name to ID mappings. If you change a filename outside afpd(8) (shell, samba), the CNID db, i.e. the DIDNAME index, gets inconsistent. Netatalk tries to recover from such inconsistencies as gracefully as possible. The mechanisms to resolve such inconsistencies may fail sometimes, though, as this is not an easy task to accomplish. I.e. if several names in the path to the file or directory have changed, things may go wrong.

If you change a lot of filenames at once, chances are higher that the afpds fallback mechanisms fail, i.e. files will be assigned new IDs, even though the file hasn't changed. **uniconv** therefore updates the CNID entry for each file/directory directly after it changes the name to avoid inconsistencies. The two supported backends for volumes, dbd and cdb, use the same CNID db format. Therefore, you *could* use **uniconv** with cdb and **afpd** with dbd later.

Warning: There must not be two processes opening the CNID database using different backends at once! If a volume is still opened with dbd (cnid_metad/cnid_dbd) and you start **uniconv** with cdb, the result will be a corrupted CNID database, as the two backends use different locking schemes. You might run into additional problems, e.g. if dbd is compiled with transactions, cdb will not update the transaction logs.

In general, it is recommended to use the same backend for **uniconv** you are using with afpd(8).

Examples

convert 1.x CAP encoded volume to UTF-8, clients used MacRoman codepage, cnidscheme is dbd:

```
example% uniconv -c dbd -f ASCII -t UTF8 -m MAC_ROMAN /path/to/share
```

convert iso8859-1 volume to UTF-8, cnidscheme is cdb:

```
example% uniconv -c cdb -f ISO-8859-1 -t UTF8 -m MAC_ROMAN /path/to/share
```

convert 1.x volume using iso8859-1 adapted NLS to CAP encoding:

```
example% uniconv -f ISO-8859-ADAPTED -t ASCII -m MAC_ROMAN/path/to/share
```

convert UTF-8 volume to CAP, for MacCyrillic clients:

```
example% uniconv -f UTF8 -t ASCII -m MAC_CYRILLIC /path/to/share
```

See also

AppleVolumes.default(5),afpd(8),iconv(1),cnid_metad(8),cnid_dbd(8)

Index