Amd

The 4.4 BSD Automounter

Reference Manual

Jan-Simon Pendry

and

Nick Williams

Last updated March 1991
Documentation for software revision 5.3 Alpha

Preface

This manual documents the use of the 4.4 BSD automounter—*Amd*. This is primarily a reference manual. Unfortunately, no tutorial exists.

This manual comes in two forms: the published form and the Info form. The Info form is for on-line perusal with the INFO program which is distributed along with GNU Emacs. Both forms contain substantially the same text and are generated from a common source file, which is distributed with the *Amd* source.

License

Amd is not in the public domain; it is copyrighted and there are restrictions on its distribution.

Redistribution and use in source and binary forms are permitted provided that: (1) source distributions retain this entire copyright notice and comment, and (2) distributions including binaries display the following acknowledgement: "This product includes software developed by The University of California, Berkeley and its Contributors" in the documentation or other materials provided with the distribution and in all advertising materials mentioning features or use of this software. neither the name of the University nor the names of its Contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Source Distribution

If you have access to the Internet, you can get the latest distribution version of *Amd* from host '`usc.edu`' using anonymous FTP. Move to the directory '`/pub/amd`' on that host and fetch the file '`amd.tar.Z`'.

If you are in the UK, you can get the latest distribution version of *Amd* from the UKnet info-server. Start by sending email to '`info-server@doc.ic.ac.uk`'.

Sites on the UK JANET network can get the latest distribution by using anonymous NIFTP to fetch the file '`<AMD>amd.tar.Z`' from host '`uk.ac.imperial.doc.src`'.

Revision 5.2 was part of the 4.3 BSD Reno distribution.

Revision 5.3bsdnet, a late alpha version of 5.3, was part of the BSD network version 2 distribution

Bug Reports

Send all bug reports to '`jsp@doc.ic.ac.uk`' quoting the details of the release and your configuration. These can be obtained by running the command '`amd -v`'.

Mailing List

There is a mailing list for people interested in keeping uptodate with developments. To subscribe, send a note to '`amd-workers-request@acl.lanl.gov`'.

Introduction

An *automounter* maintains a cache of mounted filesystems. Filesystems are mounted on demand when they are first referenced, and unmounted after a period of inactivity.

Amd may be used as a replacement for Sun's automounter. The choice of which filesystem to mount can be controlled dynamically with *selectors*. Selectors allow decisions of the form "hostname is *this*," or "architecture is not *that*." Selectors may be combined arbitrarily. *Amd* also supports a variety of filesystem types, including NFS, UFS and the novel *program* filesystem. The combination of selectors and multiple filesystem types allows identical configuration files to be used on all machines so reducing the administrative overhead.

Amd ensures that it will not hang if a remote server goes down. Moreover, *Amd* can determine when a remote server has become inaccessible and then mount replacement filesystems as and when they become available.

Amd contains no proprietary source code and has been ported to numerous flavours of Unix.

1. Overview

Amd maintains a cache of mounted filesystems. Filesystems are *demand-mounted* when they are first referenced, and unmounted after a period of inactivity. *Amd* may be used as a replacement for Sun's **automount**(8) program. It contains no proprietary source code and has been ported to numerous flavours of Unix. See section 2.1 [Supported Operating Systems], page 10.

Amd was designed as the basis for experimenting with filesystem layout and management. Although *Amd* has many direct applications it is loaded with additional features which have little practical use. At some point the infrequently used components may be removed to streamline the production system.

1.1 Fundamentals

The fundamental concept behind *Amd* is the ability to separate the name used to refer to a file from the name used to refer to its physical storage location. This allows the same files to be accessed with the same name regardless of where in the network the name is used. This is very different from placing '`/n/hostname`' in front of the pathname since that includes location dependent information which may change if files are moved to another machine.

By placing the required mappings in a centrally administered database, filesystems can be re-organised without requiring changes to configuration files, shell scripts and so on.

1.2 Filesystems and Volumes

Amd views the world as a set of fileservers, each containg one or more filesystems where each filesystem contains one or more *volumes*. Here the term *volume* is used to refer to a coherent set of files such as a user's home directory or a TeX distribution.

In order to access the contents of a volume, *Amd* must be told in which filesystem the volume resides and which host owns the filesystem. By default the host is assumed to be local and the volume is assumed to be the entire filesystem. If a filesystem contains more than one volume, then a *sublink* is used to refer to the sub-directory within the filesystem where the volume can be found.

1.3 Volume Naming

Volume names are defined to be unique across the entire network. A volume name is the pathname to the volume's root as known by the users of that volume. Since this name uniquely identifies the volume contents, all volumes can be named and accessed from each host, subject to administrative controls.

Volumes may be replicated or duplicated. Replicated volumes contain identical copies of the same data and reside at two or more locations in the network. Each of the replicated volumes can be used interchangeably. Duplicated volumes each have the same name but contain different, though functionally identical, data. For example, '`/vol/tex`' might be the name of a TeX distribution which varied for each machine architecture.

Amd provides facilities to take advantage of both replicated and duplicated volumes. Configuration options allow a single set of configuration data to be shared across an entire network by taking advantage of replicated and duplicated volumes.

Amd can take advantage of replacement volumes by mounting them as required should an active fileserver become unavailable.

1.4 Volume Binding

Unix implements a namespace of hierarchically mounted filesystems. Two forms of binding between names and files are provided. A *hard link* completes the binding when the name is added to the filesystem. A *soft link* delays the binding until the name is accessed. An *automounter* adds a further form in which the binding of name to filesystem is delayed until the name is accessed.

The target volume, in its general form, is a tuple (host, filesystem, sublink) which can be used to name the physical location of any volume in the network.

When a target is referenced, *Amd* ignores the sublink element and determines whether the required filesystem is already mounted. This is done by computing the local mount point for the filesystem and checking for an existing filesystem mounted at the same place. If such a filesystem already exists then it is assumed to be functionally identical to the target filesystem. By default there is a one-to-one mapping between the pair (host, filesystem) and the local mount point so this assumption is valid.

1.5 Operational Principles

Amd operates by introducing new mount points into the namespace. These are called *automount* points. The kernel sees these automount points as NFS filesystems being served by *Amd*. Having attached itself to the namespace, *Amd* is now able to control the view the rest of the system has of those mount points. RPC calls are received from the kernel one at a time.

When a *lookup* call is received *Amd* checks whether the name is already known. If it is not, the required volume is mounted. A symbolic link pointing to the volume root is then returned. Once the symbolic link is returned, the kernel will send all other requests direct to the mounted filesystem.

If a volume is not yet mounted, *Amd* consults a configuration *mount-map* corresponding to the automount point. *Amd* then makes a runtime decision on what and where to mount a filesystem based on the information obtained from the map.

Amd does not implement all the NFS requests; only those relevant to name binding such as *lookup*, *readlink* and *readdir*. Some other calls are also implemented but most simply return an error code; for example *mkdir* always returns "read-only filesystem".

1.6 Mounting a Volume

Each automount point has a corresponding mount map. The mount map contains a list of key–value pairs. The key is the name of the volume to be mounted. The value is a list of locations describing where the filesystem is stored in the network. In the source for the map the value would look like

location1 location2 … locationN

Amd examines each location in turn. Each location may contain *selectors* which control whether *Amd* can use that location. For example, the location may be restricted to use by certain hosts. Those locations which cannot be used are ignored.

Amd attempts to mount the filesystem described by each remaining location until a mount succeeds or *Amd* can no longer proceed. The latter can occur in three ways:

- If none of the locations could be used, or if all of the locations caused an error, then the last error is returned.

- If a location could be used but was being mounted in the background then *Amd* marks that mount as being "in progress" and continues with the next request; no reply is sent to the kernel.
- Lastly, one or more of the mounts may have been *deferred*. A mount is deferred if extra information is required before the mount can proceed. When the information becomes available the mount will take place, but in the mean time no reply is sent to the kernel. If the mount is deferred, *Amd* continues to try any remaining locations.

Once a volume has been mounted, *Amd* establishes a *volume mapping* which is used to satisfy subsequent requests.

1.7 Automatic Unmounting

To avoid an ever increasing number of filesystem mounts, *Amd* removes volume mappings which have not been used recently. A time-to-live interval is associated with each mapping and when that expires the mapping is removed. When the last reference to a filesystem is removed, that filesystem is unmounted. If the unmount fails, for example the filesystem is still busy, the mapping is re-instated and its time-to-live interval is extended. The global default for this grace period is controlled by the "-w" command-line option (see section 4.11 [-w Option], page 28). It is also possible to set this value on a per-mount basis (see section 3.3.4.3 [opts], page 22).

Filesystems can be forcefully timed out using the *Amq* command. See chapter 6 [Run-time Administration], page 40.

1.8 Keep-alives

Use of some filesystem types requires the presence of a server on another machine. If a machine crashes then it is of no concern to processes on that machine that the filesystem is unavailable. However, to processes on a remote host using that machine as a fileserver this event is important. This situation is most widely recognised when an NFS server crashes and the behaviour observed on client machines is that more and more processes hang. In order to provide the possibility of recovery, *Amd* implements a *keep-alive* interval timer for some filesystem types. Currently only NFS makes use of this service.

The basis of the NFS keep-alive implementation is the observation that most sites maintain replicated copies of common system data such as manual pages, most or all programs, system source code and so on. If one of those servers goes down it would be reasonable to mount one of the others as a replacement.

The first part of the process is to keep track of which fileservers are up and which are down. *Amd* does this by sending RPC requests to the servers' NFS `NullProc` and checking whether a reply is returned. While the server state is uncertain the requests are re-transmitted at three second intervals and if no reply is received after four attempts the server is marked down. If a reply is received the fileserver is marked up and stays in that state for 30 seconds at which time another NFS ping is sent.

Once a fileserver is marked down, requests continue to be sent every 30 seconds in order to determine when the fileserver comes back up. During this time any reference through *Amd* to the filesystems on that server fail with the error "Operation would block". If a replacement volume is available then it will be mounted, otherwise the error is returned to the user.

Although this action does not protect user files, which are unique on the network, or processes which do not access files via *Amd* or already have open files on the hung filesystem, it can prevent most new processes from hanging.

By default, fileserver state is not maintained for NFS/TCP mounts. The remote fileserver is always assumed to be up.

1.9 Non-blocking Operation

Since there is only one instance of *Amd* for each automount point, and usually only one instance on each machine, it is important that it is always available to service kernel calls. *Amd* goes to great lengths to ensure that it does not block in a system call. As a last resort *Amd* will fork before it attempts a system call that may block indefinitely, such as mounting an NFS filesystem. Other tasks such as obtaining filehandle information for an NFS filesystem, are done using a purpose built non-blocking RPC library which is integrated with *Amd*'s task scheduler. This library is also used to implement NFS keep-alives (see section 1.8 [Keep-alives], page 8).

Whenever a mount is deferred or backgrounded, *Amd* must wait for it to complete before replying to the kernel. However, this would cause *Amd* to block waiting for a reply to be constructed. Rather than do this, *Amd* simply *drops* the call under the assumption that the kernel RPC mechanism will automatically retry the request.

2. Supported Platforms

Amd has been ported to a wide variety of machines and operating systems. The table below lists those platforms supported by the current release.

2.1 Supported Operating Systems

The following operating systems are currently supported by *Amd*. *Amd*'s conventional name for each system is given.

`acis43` 4.3 BSD for IBM RT. Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

`aix3` AIX 3.1. Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

`aux` System V for Mac-II. Contributed by Julian Onions <jpo@cs.nott.ac.uk>

`bsd44` 4.4 BSD. Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

`concentrix`
Concentrix 5.0. Contributed by Sjoerd Mullender <sjoerd@cwi.nl>

`convex` Convex OS 7.1. Contributed by Eitan Mizrotsky <eitan@shumuji.ac.il>

`dgux` Data General DG/UX. Contributed by Mark Davies <mark@comp.vuw.ac.nz>

`fpx4` Celerity FPX 4.1/2. Contributed by Stephen Pope <scp@grizzly.acl.lanl.gov>

`hcx` Harris HCX/UX. Contributed by Chris Metcalf <metcalf@masala.lcs.mit.edu>

`hlh42` HLH OTS 1.*x* (4.2 BSD). Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

`hpux` HP-UX 6.*x* or 7.0. Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

`irix` SGI Irix. Contributed by Scott R. Presnell <srp@cgl.ucsf.edu>

`next` Mach for NeXT. Contributed by Bill Trost <trost%reed@cse.ogi.edu>

`pyrOSx` Pyramid OSx. Contributed by Stefan Petri <petri@tubsibr.UUCP>

`riscix` Acorn RISC iX. Contributed by Piete Brooks <pb@cam.cl.ac.uk>

`sos3` SunOS 3.4 & 3.5. Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

`sos4` SunOS 4.*x*. Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

`u2_2` Ultrix 2.2. Contributed by Piete Brooks <pb@cam.cl.ac.uk>

`u3_0` Ultrix 3. Contributed by Piete Brooks <pb@cam.cl.ac.uk>

`u4_0` Ultrix 4.0. Contributed by Chris Lindblad <cjl@ai.mit.edu>

`umax43` Umax 4.3 BSD. Contributed by Sjoerd Mullender <sjoerd@cwi.nl>

`utek` Utek 4.0. Contributed by Bill Trost <trost%reed@cse.ogi.edu>

`xinu43` mt Xinu MORE/bsd. Contributed by Jan-Simon Pendry <jsp@doc.ic.ac.uk>

2.2 Supported Machine Architectures

`alliant`	Alliant FX/4
`arm`	Acorn ARM
`aviion`	Data General AViiON
`encore`	Encore
`fps500`	FPS Model 500
`hp9000`	HP 9000/300 family
`hp9k8`	HP 9000/800 family
`ibm032`	IBM RT
`ibm6000`	IBM RISC System/6000
`iris4d`	SGI Iris 4D
`macII`	Apple Mac II
`mips`	MIPS RISC
`multimax`	Encore Multimax
`orion105`	HLH Orion 1/05
`sun3`	Sun-3 family
`sun4`	Sun-4 family
`tahoe`	Tahoe family
`vax`	DEC Vax

3. Mount Maps

Amd has no built-in knowledge of machines or filesystems. External *mount-maps* are used to provide the required information. Specifically, *Amd* needs to know when and under what conditions it should mount filesystems.

The map entry corresponding to the requested name contains a list of possible locations from which to resolve the request. Each location specifies filesystem type, information required by that filesystem (for example the block special device in the case of UFS), and some information describing where to mount the filesystem (see section 3.3.4.2 [fs Option], page 21). A location may also contain *selectors* (see section 3.3.3 [Selectors], page 19).

3.1 Map Types

A mount-map provides the run-time configuration information to *Amd*. Maps can be implemented in many ways. Some of the forms supported by *Amd* are regular files, ndbm databases, NIS maps the *Hesiod* name server and even the password file.

A mount-map *name* is a sequence of characters. When an automount point is created a handle on the mount-map is obtained. For each map type configured *Amd* attempts to reference the a map of the appropriate type. If a map is found, *Amd* notes the type for future use and deletes the reference, for example closing any open file descriptors. The available maps are configure when *Amd* is built and can be displayed by running the command '`amd -v`'.

By default, *Amd* caches data in a mode dependent on the type of map. This is the same as specifying '`cache:=mapdefault`' and selects a suitable default cache mode depending on the map type. The individual defaults are described below. The *cache* option can be specified on automount points to alter the caching behaviour (see section 5.8 [Automount Filesystem], page 36).

The following map types have been implemented, though some are not available on all machines. Run the command '`amd -v`' to obtain a list of map types configured on your machine.

3.1.1 File maps

When *Amd* searches a file for a map entry it does a simple scan of the file and supports both comments and continuation lines.

Continuation lines are indicated by a backslash character ('\') as the last character of a line in the file. The backslash, newline character *and any leading white space on the following line* are discarded. A maximum line length of 2047 characters is enforced after continuation lines are read but before comments are stripped. Each line must end with a newline character; that is newlines are terminators, not separators. The following examples illustrate this:

```
key     valA   valB;   \
          valC
```

specifies *three* locations, and is identical to

```
key     valA   valB;   valC
```

However,

```
key     valA   valB;\
          valC
```

specifies only *two* locations, and is identical to

```
key     valA   valB;valC
```

After a complete line has been read from the file, including continuations, *Amd* determines whether there is a comment on the line. A comment begins with a hash ("'#'") character and continues to the end of the line. There is no way to escape or change the comment lead-in character.

Note that continuation lines and comment support *only* apply to file maps, or ndbm maps built with the `mk-amd-map` program.

When caching is enabled, file maps have a default cache mode of `all` (see section 5.8 [Automount Filesystem], page 36).

3.1.2 ndbm maps

An ndbm map may be used as a fast access form of a file map. The program, `mk-amd-map`, converts a normal map file into an ndbm database. This program supports the same continuation and comment conventions that are provided for file maps. Note that ndbm format files may *not* be sharable across machine architectures. The notion of speed generally only applies to large maps; a small map, less than a single disk block, is almost certainly better implemented as a file map.

ndbm maps do not support cache mode '`all`' and, when caching is enabled, have a default cache mode of '`inc`' (see section 5.8 [Automount Filesystem], page 36).

3.1.3 NIS maps

When using NIS (formerly YP), an *Amd* map is implemented directly by the underlying NIS map. Comments and continuation lines are *not* supported in the automounter and must be stripped when constructing the NIS server's database.

NIS maps do not support cache mode `all` and, when caching is enabled, have a default cache mode of `inc` (see section 5.8 [Automount Filesystem], page 36).

The following rule illustrates what could be added to your NIS '`Makefile`', in this case causing the '`amd.home`' map to be rebuilt:

```
$(YPTSDIR)/amd.home.time: $(ETCDIR)/amd.home
        -@sed -e "s/#.*$$//" -e "/^$$/d" $(ETCDIR)/amd.home | \
          awk '{  \
                 for (i = 1; i <= NF; i++) \
                     if (i == NF) { \
                         if (substr($$i, length($$i), 1) == "\\") \
                             printf("%s", substr($$i, 1, length($$i) - 1)); \
                         else \
                             printf("%s\n", $$i); \
                     } \
                     else \
                         printf("%s ", $$i); \
             }' | \
        $(MAKEDBM) - $(YPDBDIR)/amd.home; \
        touch $(YPTSDIR)/amd.home.time; \
        echo "updated amd.home"; \
        if [ ! $(NOPUSH) ]; then \
                $(YPPUSH) amd.home; \
                echo "pushed amd.home"; \
        else \
                : ; \
        fi
```

Here `$(YPTSDIR)` contains the time stamp files, and `$(YPDBDIR)` contains the dbm format NIS files.

3.1.4 Hesiod maps

When the map name begins with the string '`hesiod.`' lookups are made using the *Hesiod* name server. The string following the dot is used as a name qualifier and is prepended with the key being located. The entire string is then resolved in the `automount` context. For example, if the the key is '`jsp`' and map name is '`hesiod.homes`' then *Hesiod* is asked to resolve '`jsp.homes.automount`'.

Hesiod maps do not support cache mode '`all`' and, when caching is enabled, have a default cache mode of '`inc`' (see section 5.8 [Automount Filesystem], page 36).

The following is an example of a *Hesiod* map entry:

```
jsp.homes.automount HS TXT "rfs:=/home/charm;rhost:=charm;sublink:=jsp"
njw.homes.automount HS TXT "rfs:=/home/dylan/dk2;rhost:=dylan;sublink:=njw"
```

3.1.5 Password maps

The password map support is unlike the four previous map types. When the map name is the string '`/etc/passwd`' *Amd* can lookup a user name in the password file and re-arrange the home directory field to produce a usable map entry.

Amd assumes the home directory has the format '/*anydir*/*dom1*/../*domN*/*login*'. It breaks this string into a map entry where `${rfs}` has the value '/*anydir*/*domN*', `${rhost}` has the value '*domN*.....*dom1*', and `${sublink}` has the value '`login`'.

Thus if the password file entry was

```
/home/achilles/jsp
```

the map entry used by *Amd* would be

```
rfs:=/home/achilles;rhost:=achilles;sublink:=jsp
```

Similarly, if the password file entry was

```
/home/cc/sugar/mjh
```

the map entry used by *Amd* would be

```
rfs:=/home/sugar;rhost:=sugar.cc;sublink:=jsp
```

3.1.6 Union maps

The union map support is provided specifically for use with the union filesystem, see section 5.10 [Union Filesystem], page 38.

It is identified by the string '`union:`' which is followed by a colon separated list of directories. The directories are read in order, and the names of all entries are recorded in the map cache. Later directories take precedence over earlier ones. The union filesystem type then uses the map cache to determine the union of the names in all the directories.

3.2 How keys are looked up

The key is located in the map whose type was determined when the automount point was first created. In general the key is a pathname component. In some circumstances this may be modified by variable expansion (see section 3.3.2 [Variable Expansion], page 18) and prefixing. If the automount point has a prefix, specified by the *pref* option, then that is prepended to the search key before the map is searched.

If the map cache is a '`regexp`' cache then the key is treated as an egrep-style regular expression, otherwise a normal string comparison is made.

If the key cannot be found then a *wildcard* match is attempted. *Amd* repeatedly strips the basename from the key, appends '`/*`' and attempts a lookup. Finally, *Amd* attempts to locate the special key '`*`'.

For example, the following sequence would be checked if '`home/dylan/dk2`' was being located:

```
home/dylan/dk2
home/dylan/*
home/*
*
```

At any point when a wildcard is found, *Amd* proceeds as if an exact match had been found and the value field is then used to resolve the mount request, otherwise an error code is propagated back to the kernel. (see chapter 5 [Filesystem Types], page 31).

3.3 Location Format

The value field from the lookup provides the information required to mount a filesystem. The information is parsed according to the syntax shown below.

location-list:
  location-selection
  location-list white-space `||` *white-space location-selection*
location-selection:
  location
  location-selection white-space location
location:
  location-info
  `-`*location-info*
  `-`
location-info:
  sel-or-opt
  location-info`;`*sel-or-opt*
  `;`
sel-or-opt:
  selection
  opt-ass
selection:
  selector`==`*value*
  selector`!=`*value*
opt-ass:
  option`:=`*value*
white-space:
  space
  tab

Note that unquoted whitespace is not allowed in a location description. White space is only allowed, and is mandatory, where shown with non-terminal '`white-space`'.

A *location-selection* is a list of possible volumes with which to satisfy the request. *location-selections* are separated by the '`||`' operator. The effect of this operator is to prevent use of location-selections to its right if any of the location-selections on its left were selected whether or not any of them were successfully mounted (see section 3.3.3 [Selectors], page 19).

The location-selection, and singleton *location-list*, '`type:=ufs;dev:=/dev/xd1g`' would inform *Amd* to mount a UFS filesystem from the block special device '`/dev/xd1g`'.

The *sel-or-opt* component is either the name of an option required by a specific filesystem, or it is the name of a built-in, predefined selector such as the architecture type. The value may be quoted with double quotes '`"`', for example '`type:="ufs";dev:="/dev/xd1g"`'. These quotes are stripped when the value is parsed and there is no way to get a double quote into a value field. Double quotes are used to get white space into a value field, which is needed for the program filesystem (see section 5.5 [Program Filesystem], page 34).

3.3.1 Map Defaults

A location beginning with a dash '`-`' is used to specify default values for subsequent locations. Any previously specified defaults in the location-list are discarded. The default string can be empty in which case no defaults apply.

The location '`-fs:=/mnt;opts:=ro`' would set the local mount point to '`/mnt`' and cause mounts to be read-only by default. Defaults specified this way are appended to, and so override, any global map defaults given with '`/defaults`').

3.3.2 Variable Expansion

To allow generic location specifications *Amd* does variable expansion on each location and also on some of the option strings. Any option or selector appearing in the form `$`*var* is replaced by the current value of that option or selector. For example, if the value of `${key}` was '`bin`', `${autodir}` was '`/a`' and `${fs}` was '`${autodir}/local/${key}`' then after expansion `${fs}` would have the value '`/a/local/bin`'. Any environment variable can be accessed in a similar way.

Two pathname operators are available when expanding a variable. If the variable name begins with '`/`' then only the last component of then pathname is substituted. For example, if `${path}` was '`/foo/bar`' then `${/path}` would be expanded to '`bar`'. Similarly, if the variable name ends with '`/`' then all but the last component of the pathname is substituted. In the previous example, `${path/}` would be expanded to '`/foo`'.

Two domain name operators are also provided. If the variable name begins with '`.`' then only the domain part of the name is substituted. For example, if `${rhost}` was '`swan.doc.ic.ac.uk`' then `${.rhost}` would be expanded to '`doc.ic.ac.uk`'. Similarly, if the variable name ends with

'`.`' then only the host component is substituted. In the previous example, `${rhost.}` would be expanded to '`swan`'.

Variable expansion is a two phase process. Before a location is parsed, all references to selectors, *eg* `${path}`, are expanded. The location is then parsed, selections are evaluated and option assignments recorded. If there were no selections or they all succeeded the location is used and the values of the following options are expanded in the order given: *sublink*, *rfs*, *fs*, *opts*, *remopts*, *mount* and *unmount*.

Note that expansion of option values is done after *all* assignments have been completed and not in a purely left to right order as is done by the shell. This generally has the desired effect but care must be taken if one of the options references another, in which case the ordering can become significant.

There are two special cases concerning variable expansion:

1. before a map is consulted, any selectors in the name received from the kernel are expanded. For example, if the request from the kernel was for '`${arch}.bin`' and the machine architecture was '`vax`', the value given to `${key}` would be '`vax.bin`'.
2. the value of `${rhost}` is expanded and normalized before the other options are expanded. The normalization process strips any local sub-domain components. For example, if `${domain}` was '`Berkeley.EDU`' and `${rhost}` was initially '`snow.Berkeley.EDU`', after the normalization it would simply be '`snow`'. Hostname normalization is currently done in a *case-dependent* manner.

3.3.3 Selectors

Selectors are used to control the use of a location. It is possible to share a mount map between many machines in such a way that filesystem location, architecture and operating system differences are hidden from the users. A selector of the form '`arch==sun3;os==sos4`' would only apply on Sun-3s running SunOS 4.x.

Selectors are evaluated left to right. If a selector fails then that location is ignored. Thus the selectors form a conjunction and the locations form a disjunction. If all the locations are ignored or otherwise fail then *Amd* uses the *error* filesystem (see section 5.11 [Error Filesystem], page 38). This is equivalent to having a location '`type:=error`' at the end of each mount-map entry.

The selectors currently implemented are:

'arch' the machine architecture which was automatically determined at compile time. The architecture type can be displayed by running the command '`amd -v`'. See section 2.2 [Supported Machine Architectures], page 11.

'autodir' the default directory under which to mount filesystems. This may be changed by the "-a" command line option. See the *fs* option.

'byte' the machine's byte ordering. This is either '`little`', indicating little-endian, or '`big`', indicating big-endian. One possible use is to share '`rwho`' databases (see section 8.5 [rwho servers], page 69). Another is to share ndbm databases, however this use can be considered a courageous juggling act.

'cluster' is provided as a hook for the name of the local cluster. This can be used to decide which servers to use for copies of replicated filesystems. `${cluster}` defaults to the value of `${domain}` unless a different value is set with the "-C" command line option.

'domain' the local domain name as specified by the "-d" command line option. See '`host`'.

'host' the local hostname as determined by **gethostname**(2). If no domain name was specified on the command line and the hostname contains a period '`.`' then the string before the period is used as the host name, and the string after the period is assigned to `${domain}`. For example, if the hostname is '`styx.doc.ic.ac.uk`' then `host` would be '`styx`' and `domain` would be '`doc.ic.ac.uk`'. `hostd` would be '`styx.doc.ic.ac.uk`'.

'hostd' is `${host}` and `${domain}` concatenated with a '`.`' inserted between them if required. If `${domain}` is an empty string then `${host}` and `${hostd}` will be identical.

'karch' is provided as a hook for the kernel architecture. This is used on SunOS 4, for example, to distinguish between different '`/usr/kvm`' volumes. `${karch}` defaults to the value of `${arch}` unless a different value is set with the "-k" command line option.

'os' the operating system. Like the machine architecture, this is automatically determined at compile time. The operating system name can be displayed by running the command '`amd -v`'. See section 2.1 [Supported Operating Systems], page 10.

The following selectors are also provided. Unlike the other selectors, they vary for each lookup. Note that when the name from the kernel is expanded prior to a map lookup, these selectors are all defined as empty strings.

'key' the name being resolved. For example, if '`/home`' is an automount point, then accessing '`/home/foo`' would set `${key}` to the string '`foo`'. The key is prefixed by the *pref* option set in the parent mount point. The default prefix is an empty string. If the prefix was '`blah/`' then `${key}` would be set to '`blah/foo`'.

'map' the name of the mount map being used.

'path' the full pathname of the name being resolved. For example '`/home/foo`' in the example above.

'wire' the name of the network to which the primary network interface is attached. If a symbolic name cannot be found in the networks or hosts database then dotted IP address format is used. This value is also output by the "-v" option.

Selectors can be negated by using '!=' instead of '=='. For example to select a location on all non-Vax machines the selector 'arch!=vax' would be used.

3.3.4 Map Options

Options are parsed concurrently with selectors. The difference is that when an option is seen the string following the ':=' is recorded for later use. As a minimum the *type* option must be specified. Each filesystem type has other options which must also be specified. See chapter 5 [Filesystem Types], page 31, for details on the filesystem specific options.

Superfluous option specifications are ignored and are not reported as errors.

The following options apply to more than one filesystem type.

3.3.4.1 delay Option

The delay, in seconds, before an attempt will be made to mount from the current location. Auxilliary data, such as network address, file handles and so on are computed regardless of this value.

A delay can be used to implement the notion of primary and secondary file servers. The secondary servers would have a delay of a few seconds, thus giving the primary servers a chance to respond first.

3.3.4.2 fs Option

The local mount point. The semantics of this option vary between filesystems.

For NFS and UFS filesystems the value of `${fs}` is used as the local mount point. For other filesystem types it has other meanings which are described in the section describing the respective filesystem type. It is important that this string uniquely identifies the filesystem being mounted. To satisfy this requirement, it should contain the name of the host on which the filesystem is resident and the pathname of the filesystem on the local or remote host.

The reason for requiring the hostname is clear if replicated filesystems are considered. If a fileserver goes down and a replacement filesystem is mounted then the *local* mount point *must* be different from that of the filesystem which is hung. Some encoding of the filesystem name is required if more than one filesystem is to be mounted from any given host.

If the hostname is first in the path then all mounts from a particular host will be gathered below a single directory. If that server goes down then the hung mount points are less likely to be accidentally referenced, for example when **getwd**(3) traverses the namespace to find the pathname of the current directory.

The '`fs`' option defaults to `${autodir}/${rhost}${rfs}`. In addition, '`rhost`' defaults to the local host name (`${host}`) and '`rfs`' defaults to the value of `${path}`, which is the full path of the requested file; '`/home/foo`' in the example above (see section 3.3.3 [Selectors], page 19). `${autodir}` defaults to '`/a`' but may be changed with the "-a" command line option. Sun's automounter defaults to '`/tmp_mnt`'. Note that there is no '`/`' between the `${rhost}` and `${rfs}` since `${rfs}` begins with a '`/`'.

3.3.4.3 opts Option

The options to pass to the mount system call. A leading '`-`' is silently ignored. The mount options supported generally correspond to those used by **mount**(8) and are listed below. Some additional pseudo-options are interpreted by *Amd* and are also listed.

Unless specifically overridden, each of the system default mount options applies. Any options not recognised are ignored. If no options list is supplied the string '`rw,defaults`' is used and all the system default mount options apply. Options which are not applicable for a particular operating system are silently ignored. For example, only 4.4 BSD is known to implement the `compress` and `spongy` options.

`compress` Use NFS compression protocol.

`grpid` Use BSD directory group-id semantics.

`intr` Allow keyboard interrupts on hard mounts.

`noconn` Don't make a connection on datagram transports.

`nocto` No close-to-open consistency.

`nodevs` Don't allow local special devices on this filesystem.

`nolooklease`
: When nqnfs is selected, disable the automatic acquiring of leases on lookups.

`nosuid` Don't allow set-uid or set-gid executables on this filesystem.

`nqnfs` Enable leasing extensions to the NFS protocol know as Not Quite NFS. Only supported by the 4.4BSD NFS implementation.

`quota` Enable quota checking on this mount.

`resvport` Use a reserved port number (one less than 1024). Some NFS servers require this.

`retrans=`*n*
: The number of NFS retransmits made before a user error is generated by a '`soft`' mounted filesystem, and before a '`hard`' mounted filesystem reports '`NFS server` *yoyo* `not responding still trying`'.

`ro` Mount this filesystem readonly.

`rsize=`*n* The NFS read packet size. You may need to set this if you are using NFS/UDP through a gateway.

`soft` Give up after *retrans* retransmissions.

`spongy` Like '`soft`' for status requests, and '`hard`' for data transfers.

`tcp` Use TCP/IP instead of UDP/IP, ignored if the NFS implementation does not support TCP/IP mounts.

`timeo=`*n* The NFS timeout, in tenth-seconds, before a request is retransmitted.

`wsize=`*n* The NFS write packet size. You may need to set this if you are using NFS/UDP through a gateway.

The following options are implemented by *Amd*, rather than being passed to the kernel.

`nounmount`
: Configures the mount so that its time-to-live will never expire. This is also the default for some filesystem types.

`ping=`*n* The interval, in seconds, between keep-alive pings. When four consecutive pings have failed the mount point is marked as hung. This interval defaults to 30 seconds. If the ping interval is less than zero, no pings are sent and the host is assumed to be always up. By default, pings are not sent for an NFS/TCP mount.

`retry=`*n* The number of times to retry the mount system call.

`utimeout=`*n*
: The interval, in seconds, by which the mount's time-to-live is extended after an unmount attempt has failed. In fact the interval is extended before the unmount is attempted to avoid thrashing. The default value is 120 seconds (two minutes) or as set by the "-w" command line option.

3.3.4.4 remopts Option

This option has the same use as `${opts}` but applies only when the remote host is on a non-local network. For example, when using NFS across a gateway it is often necessary to use smaller values for the data read and write sizes. This can simply be done by specifying the small values in *remopts*. When a non-local host is accessed, the smaller sizes will automatically be used.

Amd determines whether a host is local by examining the network interface configuration at startup. Any interface changes made after *Amd* has been started will not be noticed. The likely effect will be that a host may incorrectly be declared non-local.

Unless otherwise set, the value of `${rem}` is the same as the value of `${opts}`.

3.3.4.5 sublink Option

The subdirectory within the mounted filesystem to which the reference should point. This can be used to prevent duplicate mounts in cases where multiple directories in the same mounted filesystem are used.

3.3.4.6 type Option

The filesystem type to be used. See chapter 5 [Filesystem Types], page 31, for a full description of each type.

4. *Amd* Command Line Options

Many of *Amd*'s parameters can be set from the command line. The command line is also used to specify automount points and maps.

The general format of a command line is

`amd [`*options*`] { `*directory map-name* `[-`*map-options*`] } ...`

For each directory and map-name given, *Amd* establishes an automount point. The *map-options* may be any sequence of options or selectors—see section 3.3 [Location Format], page 17. The *map-options* apply only to *Amd*'s mount point.

'`type:=toplvl;cache:=mapdefault;fs:=${map}`' is the default value for the map options. Default options for a map are read from a special entry in the map whose key is the string '`/defaults`'. When default options are given they are prepended to any options specified in the mount-map locations as explained in. See section 3.3.1 [Map Defaults], page 18, for more details.

The *options* are any combination of those listed below.

Once the command line has been parsed, the automount points are mounted. The mount points are created if they do not already exist, in which case they will be removed when *Amd* exits. Finally, *Amd* disassociates itself from its controlling terminal and forks into the background.

Note: Even if *Amd* has been built with '`-DDEBUG`' it will still background itself and disassociate itself from the controlling terminal. To use a debugger it is necessary to specify '`-D nodaemon`' on the command line.

4.1 `-a` *directory*

Specifies the default mount directory. This option changes the variable `${autodir}` which otherwise defaults to '`/a`'. For example, some sites prefer '`/amd`'.

```
amd -a /amd ...
```

4.2 `-c` *cache-interval*

Selects the period, in seconds, for which a name is cached by *Amd*. If no reference is made to the volume in this period, *Amd* discards the volume name to filesystem mapping.

Once the last reference to a filesystem has been removed, *Amd* attempts to unmount the filesystem. If the unmount fails the interval is extended by a further period as specified by the '`-w`' command line option or by the '`utimeout`' mount option.

The default *cache-interval* is 300 seconds (five minutes).

4.3 `-d` *domain*

Specifies the host's domain. This sets the internal variable `${domain}` and affects the `${hostd}` variable.

If this option is not specified and the hostname already contains the local domain then that is used, otherwise the default value of `${domain}` is '`unknown.domain`'.

For example, if the local domain was '`doc.ic.ac.uk`', *Amd* could be started as follows:

```
amd -d doc.ic.ac.uk ...
```

4.4 `-k` *kernel-architecture*

Specifies the kernel architecture of the system. This is usually the output of '`arch -k`' and its only effect is to set the variable `${karch}`. If this option is not given, `${karch}` has the same value as `${arch}`.

This would be used as follows:

```
amd -k `arch -k` ...
```

4.5 `-l` *log-option*

Selects the form of logging to be made. Two special *log-options* are recognised.

1. If *log-option* is the string '`syslog`', *Amd* will use the **syslog**(3) mechanism.
2. If *log-option* is the string '`/dev/stderr`', *Amd* will use standard error, which is also the default target for log messages. To implement this, *Amd* simulates the effect of the '`/dev/fd`' driver.

Any other string is taken as a filename to use for logging. Log messages are appended to the file if it already exists, otherwise a new file is created. The file is opened once and then held open, rather than being re-opened for each message.

If the '`syslog`' option is specified but the system does not support syslog or if the named file cannot be opened or created, *Amd* will use standard error. Error messages generated before *Amd* has finished parsing the command line are printed on standard error.

Using '`syslog`' is usually best, in which case *Amd* would be started as follows:

```
amd -l syslog ...
```

4.6 `-n`

Normalises the remote hostname before using it. Normalisation is done by replacing the value of `${rhost}` with the primary name returned by a hostname lookup.

This option should be used if several names are used to refer to a single host in a mount map.

4.7 `-p`

Causes *Amd*'s process id to be printed on standard output. This can be redirected to a suitable file for use with kill:

```
amd -p > /var/run/amd.pid ...
```

This option only has an affect if *Amd* is running in daemon mode. If *Amd* is started with the `-D nodaemon` debug flag, this option is ignored.

4.8 `-r`

Tells *Amd* to restart existing mounts (see section 5.14 [Inheritance Filesystem], page 39).

4.9 `-t` *timeout.retransmit*

Specifies the RPC *timeout* and *retransmit* intervals used by the kernel to communicate to *Amd*. These are used to set the '`timeo`' and '`retrans`' mount options.

Amd relies on the kernel RPC retransmit mechanism to trigger mount retries. The value of this parameter changes the retry interval. Too long an interval gives poor interactive response, too short an interval causes excessive retries.

4.10 `-v`

Print version information on standard error and then exit. The output is of the form:

```
amd 5.2.1.11 of 91/03/17 18:04:05 5.3Alpha11 #0: Sun Mar 17 18:07:28 GMT 1991
Built by pendry@vangogh.Berkeley.EDU for a hp300 running bsd44 (big-endian).
Map support for: root, passwd, union, file, error.
FS: ufs, nfs, nfsx, host, link, program, union, auto, direct, toplvl, error.
Primary network is 128.32.130.0.
```

The information includes the version number, release date and name of the release. The architecture (see section 2.2 [Supported Machine Architectures], page 11), operating system (see section 2.1 [Supported Operating Systems], page 10) and byte ordering are also printed as they appear in the `${os}`, `${arch}` and `${byte}` variables.

4.11 `-w` *wait-timeout*

Selects the interval in seconds between unmount attempts after the initial time-to-live has expired.

This defaults to 120 seconds (two minutes).

4.12 `-x` *opts*

Specifies the type and verbosity of log messages. *opts* is a comma separated list selected from the following options:

`fatal`	Fatal errors
`error`	Non-fatal errors
`user`	Non-fatal user errors
`warn`	Recoverable errors
`warning`	Alias for `warn`
`info`	Information messages
`map`	Mount map usage
`stats`	Additional statistics
`all`	All of the above

Initially a set of default logging flags is enabled. This is as if '`-x all,nomap,nostats`' had been selected. The command line is parsed and logging is controlled by the "-x" option. The very first set of logging flags is saved and can not be subsequently disabled using *Amq*. This default set of options is useful for general production use.

The '`info`' messages include details of what is mounted and unmounted and when filesystems have timed out. If you want to have the default set of messages without the '`info`' messages then you simply need '`-x noinfo`'. The messages given by '`user`' relate to errors in the mount maps, so these are useful when new maps are installed. The following table lists the syslog priorites used for each of the message types.

`fatal`	LOG_CRIT
`error`	LOG_ERR
`user`	LOG_WARNING
`warning`	LOG_WARNING
`info`	LOG_INFO
`debug`	LOG_DEBUG
`map`	LOG_DEBUG

`stats` LOG_INFO

The options can be prefixed by the string '`no`' to indicate that this option should be turned off. For example, to obtain all but '`info`' messages the option '`-x all,noinfo`' would be used.

If *Amd* was built with debugging enabled the `debug` option is automatically enabled regardless of the command line options.

4.13 `-y` *NIS-domain*

Selects an alternate NIS domain. This is useful for debugging and cross-domain shared mounting. If this flag is specified, *Amd* immediately attempts to bind to a server for this domain.

4.14 `-C` *cluster-name*

Specifies the name of the cluster of which the local machine is a member. The only effect is to set the variable `${cluster}`. The *cluster-name* is will usually obtained by running another command which uses a database to map the local hostname into a cluster name. `${cluster}` can then be used as a selector to restrict mounting of replicated data. If this option is not given, `${cluster}` has the same value as `${domain}`. This would be used as follows:

```
amd -C `clustername` ...
```

4.15 `-D` *opts*

Controls the verbosity and coverage of the debugging trace; *opts* is a comma separated list of debugging options. The "-D" option is only available if *Amd* was compiled with '`-DDEBUG`'. The memory debugging facilities are only available if *Amd* was compiled with '`-DDEBUG_MEM`' (in addition to '`-DDEBUG`').

The most common options to use are '`-D trace`' and '`-D test`' (which turns on all the useful debug options). See the program source for a more detailed explanation of the available options.

5. Filesystem Types

To mount a volume, *Amd* must be told the type of filesystem to be used. Each filesystem type typically requires additional information such as the fileserver name for NFS.

From the point of view of *Amd*, a *filesystem* is anything that can resolve an incoming name lookup. An important feature is support for multiple filesystem types. Some of these filesystems are implemented in the local kernel and some on remote fileservers, whilst the others are implemented internally by *Amd*.

The two common filesystem types are UFS and NFS. Four other user accessible filesystems (‘`link`’, ‘`program`’, ‘`auto`’ and ‘`direct`’) are also implemented internally by *Amd* and these are described below. There are two additional filesystem types internal to *Amd* which are not directly accessible to the user (‘`inherit`’ and ‘`error`’). Their use is described since they may still have an effect visible to the user.

5.1 Network Filesystem (‘`type:=nfs`’)

The *nfs* filesystem type provides access to Sun’s NFS.

The following options must be specified:

`rhost` the remote fileserver. This must be an entry in the hosts database. IP addresses are not accepted. The default value is taken from the local host name (`${host}`) if no other value is specified.

`rfs` the remote filesystem. If no value is specified for this option, an internal default of `${path}` is used.

NFS mounts require a two stage process. First, the *file handle* of the remote file system must be obtained from the server. Then a mount system call must be done on the local system. *Amd* keeps a cache of file handles for remote file systems. The cache entries have a lifetime of a few minutes.

If a required file handle is not in the cache, *Amd* sends a request to the remote server to obtain it. *Amd does not* wait for a response; it notes that one of the locations needs retrying, but continues with any remaining locations. When the file handle becomes available, and assuming none of the other locations was successfully mounted, *Amd* will retry the mount. This mechanism allows several

NFS filesystems to be mounted in parallel. The first one which responds with a valid file handle will be used.

An NFS entry might be:

```
jsp  host!=charm;type:=nfs;rhost:=charm;rfs:=/home/charm;sublink:=jsp
```

The mount system call and any unmount attempts are always done in a new task to avoid the possibilty of blocking *Amd*.

5.2 Network Host Filesystem ('`type:=host`')

The *host* filesystem allows access to the entire export tree of an NFS server. The implementation is layered above the '`nfs`' implementation so keep-alives work in the same way. The only option which needs to specified is '`rhost`' which is the name of the fileserver to mount.

The '`host`' filesystem type works by querying the mount daemon on the given fileserver to obtain its export list. *Amd* then obtains filehandles for each of the exported filesystems. Any errors at this stage cause that particular filesystem to be ignored. Finally each filesystem is mounted. Again, errors are logged but ignored. One common reason for mounts to fail is that the mount point does not exist. Although *Amd* attempts to automatically create the mount point, it may be on a remote filesystem to which *Amd* does not have write permission.

When an attempt to unmount a '`host`' filesystem mount fails, *Amd* remounts any filesystems which had successfully been unmounted. To do this *Amd* queries the mount daemon again and obtains a fresh copy of the export list. *Amd* then tries to mount any exported filesystems which are not currently mounted.

Sun's automounter provides a special '`-hosts`' map. To achieve the same effect with *Amd* requires two steps. First a mount map must be created as follows:

```
/defaults  type:=host;fs:=${autodir}/${rhost}/root;rhost:=${key}
*          opts:=rw,nosuid,grpid
```

and then start *Amd* with the following command

```
amd /n net.map
```

where '`net.map`' is the name of map described above. Note that the value of `${fs}` is overridden in the map. This is done to avoid a clash between the mount tree and any other filesystem already mounted from the same fileserver.

If different mount options are needed for different hosts then additional entries can be added to the map, for example

```
host2       opts:=ro,nosuid,soft
```

would soft mount '`host2`' read-only.

5.3 Network Filesystem Group ('`type:=nfsx`')

The *nfsx* filesystem allows a group of filesystems to be mounted from a single NFS server. The implementation is layered above the '`nfs`' implementation so keep-alives work in the same way.

The options are the same as for the '`nfs`' filesystem with one difference.

The following options must be specified:

`rhost` the remote fileserver. This must be an entry in the hosts database. IP addresses are not accepted. The default value is taken from the local host name (`${host}`) if no other value is specified.

`rfs` as a list of filesystems to mount. The list is in the form of a comma separated strings.

For example:

```
pub      type:=nfsx;rhost:=gould;\
rfs:=/public,/,graphics,usenet;fs:=${autodir}/${rhost}/root
```

The first string defines the root of the tree, and is applied as a prefix to the remaining members of the list which define the individual filesystems. The first string is *not* used as a filesystem name. A parallel operation is used to determine the local mount points to ensure a consistent layout of a tree of mounts.

Here, the *three* filesystems, '`/public`', '`/public/graphics`' and '`/public/usenet`', would be mounted.

A local mount point, `${fs}`, *must* be specified. The default local mount point will not work correctly in the general case. A suggestion is to use '`fs:=${autodir}/${rhost}/root`'.

5.4 Unix Filesystem ('`type:=ufs`')

The *ufs* filesystem type provides access to the system's standard disk filesystem—usually a derivative of the Berkeley Fast Filesystem.

The following option must be specified:

`dev` the block special device to be mounted.

A UFS entry might be:

```
jsp   host==charm;type:=ufs;dev:=/dev/xd0g;sublink:=jsp
```

5.5 Program Filesystem ('`type:=program`')

The *program* filesystem type allows a program to be run whenever a mount or unmount is required. This allows easy addition of support for other filesystem types, such as MIT's Remote Virtual Disk (RVD) which has a programmatic interface via the commands '`rvdmount`' and '`rvdunmount`'.

The following options must be specified:

`mount` the program which will perform the mount.

`unmount` the program which will perform the unmount.

The exit code from these two programs is interpreted as a Unix error code. As usual, exit code zero indicates success. To execute the program *Amd* splits the string on whitespace to create an array of substrings. Single quotes '`''`' can be used to quote whitespace if that is required in an argument. There is no way to escape or change the quote character.

To run the program '`rvdmount`' with a host name and filesystem as arguments would be specified by '`mount:="/etc/rvdmount rvdmount fserver ${path}"`'.

The first element in the array is taken as the pathname of the program to execute. The other members of the array form the argument vector to be passed to the program, *including argument zero*. This means that the split string must have at least two elements. The program is directly executed by *Amd*, not via a shell. This means that scripts must begin with a `#!` interpreter specification.

If a filesystem type is to be heavily used, it may be worthwhile adding a new filesystem type into *Amd*, but for most uses the program filesystem should suffice.

When the program is run, standard input and standard error are inherited from the current values used by *Amd*. Standard output is a duplicate of standard error. The value specified with the "-l" command line option has no effect on standard error.

5.6 Symbolic Link Filesystem ('`type:=link`')

Each filesystem type creates a symbolic link to point from the volume name to the physical mount point. The '`link`' filesystem does the same without any other side effects. This allows any part of the machines name space to be accessed via *Amd*.

One common use for the symlink filesystem is '`/homes`' which can be made to contain an entry for each user which points to their (auto-mounted) home directory. Although this may seem rather expensive, it provides a great deal of administrative flexibility.

The following option must be defined:

`fs` The value of *fs* option specifies the destination of the link, as modified by the *sublink* option. If *sublink* is non-null, it is appended to `${fs}/` and the resulting string is used as the target.

The '`link`' filesystem can be though of as identical to the '`ufs`' filesystem but without actually mounting anything.

An example entry might be:

```
jsp   host==charm;type:=link;fs:=/home/charm;sublink:=jsp
```

which would return a symbolic link pointing to '`/home/charm/jsp`'.

5.7 Symbolic Link Filesystem II ('type:=link')

The '`linkx`' filesystem type is identical to '`link`' with the exception that the target of the link must exist. Existence is checked with the '`lstat`' system call.

The '`linkx`' filesystem type is particularly useful for wildcard map entries. In this case, a list of possible targets can be give and *Amd* will choose the first one which exists on the local machine.

5.8 Automount Filesystem ('type:=auto')

The *auto* filesystem type creates a new automount point below an existing automount point. Top-level automount points appear as system mount points. An automount mount point can also appear as a sub-directory of an existing automount point. This allows some additional structure to be added, for example to mimic the mount tree of another machine.

The following options may be specified:

`cache` specifies whether the data in this mount-map should be cached. The default value is '`none`', in which case no caching is done in order to conserve memory. However, better performance and reliability can be obtained by caching some or all of a mount-map.

If the cache option specifies '`all`', the entire map is enumerated when the mount point is created.

If the cache option specifies '`inc`', caching is done incrementally as and when data is required. Some map types do not support cache mode '`all`', in which case '`inc`' is used whenever '`all`' is requested.

Caching can be entirely disabled by using cache mode '`none`'.

If the cache option specifies '`regexp`' then the entire map will be enumerated and each key will be treated as an egrep-style regular expression. The order in which a cached map is searched does not correspond to the ordering in the source map so the regular expressions should be mutually exclusive to avoid confusion.

Each mount map type has a default cache type, usually '`inc`', which can be selected by specifying '`mapdefault`'.

The cache mode for a mount map can only be selected on the command line. Starting *Amd* with the command:

```
amd /homes hesiod.homes -cache:=inc
```

will cause '`/homes`' to be automounted using the *Hesiod* name server with local incremental caching of all successfully resolved names.

All cached data is forgotten whenever *Amd* receives a '`SIGHUP`' signal and, if cache '`all`' mode was selected, the cache will be reloaded. This can be used to inform *Amd* that a map has been updated. In addition, whenever a cache lookup fails and *Amd* needs to examine a map, the map's modify time is examined. If the cache is out of date with respect to the map then it is flushed as if a '`SIGHUP`' had been received.

An additional option ('`sync`') may be specified to force *Amd* to check the map's modify time whenever a cached entry is being used. For example, an incremental, synchronised cache would be created by the following command:

```
amd /homes hesiod.homes -cache:=inc,sync
```

`fs` specifies the name of the mount map to use for the new mount point.

Arguably this should have been specified with the `${rfs}` option but we are now stuck with it due to historical accident.

`pref` alters the name that is looked up in the mount map. If `${pref}`, the *prefix*, is non-null then it is prepended to the name requested by the kernel *before* the map is searched.

The server '`dylan.doc.ic.ac.uk`' has two user disks: '`/dev/dsk/2s0`' and '`/dev/dsk/5s0`'. These are accessed as '`/home/dylan/dk2`' and '`/home/dylan/dk5`' respectively. Since '`/home`' is already an automount point, this naming is achieved with the following map entries:

```
dylan        type:=auto;fs:=${map};pref:=${key}/
dylan/dk2    type:=ufs;dev:=/dev/dsk/2s0
dylan/dk5    type:=ufs;dev:=/dev/dsk/5s0
```

5.9 Direct Automount Filesystem ('`type:=direct`')

The *direct* filesystem is almost identical to the automount filesystem. Instead of appearing to be a directory of mount points, it appears as a symbolic link to a mounted filesystem. The mount is done at the time the link is accessed. See section 5.8 [Automount Filesystem], page 36 for a list of required options.

Direct automount points are created by specifying the '`direct`' filesystem type on the command line:

```
amd ... /usr/man auto.direct -type:=direct
```

where '`auto.direct`' would contain an entry such as:

```
usr/man     -type:=nfs;rfs:=/usr/man \
            rhost:=man-server1  rhost:=man-server2
```

In this example, 'man-server1' and 'man-server2' are file servers which export copies of the manual pages. Note that the key which is looked up is the name of the automount point without the leading '/'.

5.10 Union Filesystem ('type:=union')

The *union* filesystem type allows the contents of several directories to be merged and made visible in a single directory. This can be used to overcome one of the major limitations of the Unix mount mechanism which only allows complete directories to be mounted.

For example, supposing '/tmp' and '/var/tmp' were to be merged into a new directory called '/mtmp', with files in '/var/tmp' taking precedence. The following command could be used to achieve this effect:

```
amd ... /mtmp union:/tmp:/var/tmp -type:=union
```

Currently, the unioned directories must *not* be automounted. That would cause a deadlock. This seriously limits the current usefulness of this filesystem type and the problem will be addressed in a future release of *Amd*.

Files created in the union directory are actually created in the last named directory. This is done by creating a wildcard entry which points to the correct directory. The wildcard entry is visible if the union directory is listed, so allowing you to see which directory has priority.

The files visible in the union directory are computed at the time *Amd* is started, and are not kept uptodate with respect to the underlying directories. Similarly, if a link is removed, for example with the 'rm' command, it will be lost forever.

5.11 Error Filesystem ('type:=error')

The *error* filesystem type is used internally as a catch-all in the case where none of the other filesystems was selected, or some other error occurred. Lookups and mounts always fail with "No such file or directory". All other operations trivially succeed.

The error filesystem is not directly accessible.

5.12 Top-level Filesystem (‘`type:=toplvl`’)

The *toplvl* filesystems is derived from the ‘`auto`’ filesystem and is used to mount the top-level automount nodes. Requests of this type are automatically generated from the command line arguments and can also be passed in by using the “-M” option of the *Amq* command.

5.13 Root Filesystem

The *root* (‘`type:=root`’) filesystem type acts as an internal placeholder onto which *Amd* can pin ‘`toplvl`’ mounts. Only one node of this type need ever exist and one is created automatically during startup. The effect of creating a second root node is undefined.

5.14 Inheritance Filesystem

The *inheritance* (‘`type:=inherit`’) filesystem is not directly accessible. Instead, internal mount nodes of this type are automatically generated when *Amd* is started with the “-r” option. At this time the system mount table is scanned to locate any filesystems which are already mounted. If any reference to these filesystems is made through *Amd* then instead of attempting to mount it, *Amd* simulates the mount and *inherits* the filesystem. This allows a new version of *Amd* to be installed on a live system simply by killing the old daemon with `SIGTERM` and starting the new one.

This filesystem type is not generally visible externally, but it is possible that the output from ‘`amq -m`’ may list ‘`inherit`’ as the filesystem type. This happens when an inherit operation cannot be completed for some reason, usually because a fileserver is down.

6. Run-time Administration

6.1 Starting *Amd*

Amd is best started from '/etc/rc.local':

```
if [ -f /etc/amd.start ]; then
        sh /etc/amd.start; (echo -n ' amd')      >/dev/console
fi
```

The shell script, 'amd.start', contains:

```
#!/bin/sh -
PATH=/etc:/bin:/usr/bin:/usr/ucb:$PATH export PATH

#
# Either name of logfile or "syslog"
#
LOGFILE=syslog
#LOGFILE=/var/log/amd

#
# Figure out whether domain name is in host name
# If the hostname is just the machine name then
# pass in the name of the local domain so that the
# hostnames in the map are domain stripped correctly.
#
case `hostname` in
*.*) dmn= ;;
*) dmn='-d doc.ic.ac.uk'
esac

#
# Zap earlier log file
#
case "$LOGFILE" in
*/*)
        mv "$LOGFILE" "$LOGFILE"-
        > "$LOGFILE"
        ;;
syslog)
        : nothing
        ;;
esac
```

```
cd /usr/sbin
#
# -r            restart
# -d dmn        local domain
# -w wait       wait between unmount attempts
# -l log        logfile or "syslog"
#
eval ./amd -r $dmn -w 240 -l "$LOGFILE" \
        /homes amd.homes -cache:=inc \
        /home amd.home -cache:=inc \
        /vol amd.vol -cache:=inc \
        /n amd.net -cache:=inc
```

If the list of automount points and maps is contained in a file or NIS map it is easily incorporated onto the command line:

```
...
eval ./amd -r $dmn -w 240 -l "$LOGFILE" `ypcat -k auto.master`
```

6.2 Stopping *Amd*

Amd stops in response to two signals.

'`SIGTERM`' causes the top-level automount points to be unmounted and then *Amd* to exit. Any automounted filesystems are left mounted. They can be recovered by restarting *Amd* with the "-r" command line option.

'`SIGINT`' causes *Amd* to attempt to unmount any filesystems which it has automounted, in addition to the actions of '`SIGTERM`'. This signal is primarily used for debugging.

Actions taken for other signals are undefined.

6.3 Controlling *Amd*

It is sometimes desirable or necessary to exercise external control over some of *Amd*'s internal state. To support this requirement, *Amd* implements an RPC interface which is used by the *Amq* program. A variety of information is available.

Amq generally applies an operation, specified by a single letter option, to a list of mount points. The default operation is to obtain statistics about each mount point. This is similar to the output shown above but includes information about the number and type of accesses to each mount point.

6.3.1 *Amq* default information

With no arguments, *Amq* obtains a brief list of all existing mounts created by *Amd*. This is different from the list displayed by **df**(1) since the latter only includes system mount points.

The output from this option includes the following information:

- the automount point,
- the filesystem type,
- the mount map or mount information,
- the internal, or system mount point.

For example:

```
/             root   "root"                     sky:(pid75)
/homes        toplvl /usr/local/etc/amd.homes  /homes
/home         toplvl /usr/local/etc/amd.home   /home
/homes/jsp    nfs    charm:/home/charm         /a/charm/home/charm/jsp
/homes/phjk   nfs    toytown:/home/toytown     /a/toytown/home/toytown/ai/phjk
```

If an argument is given then statistics for that volume name will be output. For example:

```
What            Uid   Getattr Lookup RdDir   RdLnk   Statfs Mounted@
/homes          0     1196    512    22      0       30     90/09/14 12:32:55
/homes/jsp      0     0       0      0       1180    0      90/10/13 12:56:58
```

`What` the volume name.

`Uid` ignored.

`Getattr` the count of NFS *getattr* requests on this node. This should only be non-zero for directory nodes.

`Lookup` the count of NFS *lookup* requests on this node. This should only be non-zero for directory nodes.

`RdDir` the count of NFS *readdir* requests on this node. This should only be non-zero for directory nodes.

`RdLnk` the count of NFS *readlink* requests on this node. This should be zero for directory nodes.

`Statfs` the could of NFS *statfs* requests on this node. This should only be non-zero for top-level automount points.

`Mounted@` the date and time the volume name was first referenced.

6.3.2 *Amq* -f option

The "-f" option causes *Amd* to flush the internal mount map cache. This is useful for Hesiod maps since *Amd* will not automatically notice when they have been updated. The map cache can also be synchronised with the map source by using the '`sync`' option (see section 5.8 [Automount Filesystem], page 36).

6.3.3 *Amq* -h option

By default the local host is used. In an HP-UX cluster the root server is used since that is the only place in the cluster where *Amd* will be running. To query *Amd* on another host the "-h" option should be used.

6.3.4 *Amq* -m option

The "-m" option displays similar information about mounted filesystems, rather than automount points. The output includes the following information:

- the mount information,
- the mount point,
- the filesystem type,
- the number of references to this filesystem,
- the server hostname,
- the state of the file server,
- any error which has occured.

For example:

```
"root"           truth:(pid602)     root   1 localhost is up
hesiod.home      /home              toplvl 1 localhost is up
hesiod.vol       /vol               toplvl 1 localhost is up
hesiod.homes     /homes             toplvl 1 localhost is up
amy:/home/amy    /a/amy/home/amy    nfs    5 amy is up
swan:/home/swan  /a/swan/home/swan  nfs    0 swan is up (Permission denied)
ex:/home/ex      /a/ex/home/ex      nfs    0 ex is down
```

When the reference count is zero the filesystem is not mounted but the mount point and server information is still being maintained by *Amd*.

6.3.5 *Amq* -M option

The "-M" option passes a new map entry to *Amd* and waits for it to be evaluated, possibly causing a mount. For example, the following command would cause '`/home/toytown`' on host '`toytown`' to be mounted locally on '`/mnt/toytown`'.

```
amq -M '/mnt/toytown type:=nfs;rfs:=/home/toytown;rhost:=toytown;fs:=${key}'
```

Amd applies some simple security checks before allowing this operation. The check tests whether the incoming request is from a privileged UDP port on the local machine. "Permission denied" is returned if the check fails.

A future release of *Amd* will include code to allow the **mount**(8) command to mount automount points:

```
mount -t amd /vol hesiod.vol
```

This will then allow *Amd* to be controlled from the standard system filesystem mount list.

6.3.6 *Amq* -s option

The "-s" option displays global statistics. If any other options are specified or any filesystems named then this option is ignored. For example:

```
requests  stale     mount     mount     unmount
deferred  fhandles  ok        failed    failed
1054      1         487       290       7017
```

'`Deferred requests`'
: are those for which an immediate reply could not be constructed. For example, this would happen if a background mount was required.

'`Stale filehandles`'
: counts the number of times the kernel passes a stale filehandle to *Amd*. Large numbers indicate problems.

'`Mount ok`'
: counts the number of automounts which were successful.

'`Mount failed`'
: counts the number of automounts which failed.

'`Unmount failed`'
: counts the number of times a filesystem could not be unmounted. Very large numbers here indicate that the time between unmount attempts should be increased.

6.3.7 *Amq* -u option

The "-u" option causes the time-to-live interval of the named mount points to be expired, thus causing an unmount attempt. This is the only safe way to unmount an automounted filesystem. It is not possible to unmount a filesystem which has been mounted with the '`nounmount`' flag.

6.3.8 *Amq* -v option

The "-v" option displays the version of *Amd* in a similar way to *Amd*'s "-v" option.

6.3.9 Other *Amq* options

Three other operations are implemented. These modify the state of *Amd* as a whole, rather than any particular filesystem. The "-l", "-x" and "-D" options have exactly the same effect as *Amd*'s corresponding command line options. The "-l" option is rejected by *Amd* in the current version for obvious security reasons. When *Amd* receives a "-x"flag it limits the log options being modified to those which were not enabled at startup. This prevents a user turning *off* any logging option which was specified at startup, though any which have been turned off since then can still be turned off. The "-D" option has a similar behaviour.

7. FSinfo

7.1 *FSinfo* overview

FSinfo is a filesystem management tool. It has been designed to work with *Amd* to help system administrators keep track of the ever increasing filesystem namespace under their control.

The purpose of *FSinfo* is to generate all the important standard filesystem data files from a single set of input data. Starting with a single data source guarantees that all the generated files are self-consistent. One of the possible output data formats is a set of *Amd* maps which can be used amongst the set of hosts described in the input data.

FSinfo implements a declarative language. This language is specifically designed for describing filesystem namespace and physical layouts. The basic declaration defines a mounted filesystem including its device name, mount point, and all the volumes and access permissions. *FSinfo* reads this information and builds an internal map of the entire network of hosts. Using this map, many different data formats can be produced including '`/etc/fstab`', '`/etc/exports`', *Amd* mount maps and '`/etc/bootparams`'.

7.2 Using *FSinfo*

The basic strategy when using *FSinfo* is to gather all the information about all disks on all machines into one set of declarations. For each machine being managed, the following data is required:

- Hostname
- List of all filesystems and, optionally, their mount points.
- Names of volumes stored on each filesystem.
- NFS export information for each volume.
- The list of static filesystem mounts.

The following information can also be entered into the same configuration files so that all data can be kept in one place.

- List of network interfaces

- IP address of each interface
- Hardware address of each interface
- Dumpset to which each filesystem belongs
- and more …

To generate *Amd* mount maps, the automount tree must also be defined (see section 7.8 [FSinfo automount definitions], page 56). This will have been designed at the time the volume names were allocated. Some volume names will not be automounted, so *FSinfo* needs an explicit list of which volumes should be automounted.

Hostnames are required at several places in the *FSinfo* language. It is important to stick to either fully qualified names or unqualified names. Using a mixture of the two will inevitably result in confusion.

Sometimes volumes need to be referenced which are not defined in the set of hosts being managed with *FSinfo*. The required action is to add a dummy set of definitions for the host and volume names required. Since the files generated for those particular hosts will not be used on them, the exact values used is not critical.

7.3 *FSinfo* grammar

FSinfo has a relatively simple grammar. Distinct syntactic constructs exist for each of the different types of data, though they share a common flavour. Several conventions are used in the grammar fragments below.

The notation, *list(*`xxx`*)*, indicates a list of zero or more `xxx`'s. The notation, *opt(*`xxx`*)*, indicates zero or one `xxx`. Items in double quotes, *eg* `"host"`, represent input tokens. Items in angle brackets, *eg* <*hostname*>, represent strings in the input. Strings need not be in double quotes, except to differentiate them from reserved words. Quoted strings may include the usual set of C "\" escape sequences with one exception: a backslash-newline-whitespace sequence is squashed into a single space character. To defeat this feature, put a further backslash at the start of the second line.

At the outermost level of the grammar, the input consists of a sequence of host and automount declarations. These declarations are all parsed before they are analyzed. This means they can appear in any order and cyclic host references are possible.

```
fsinfo      : list(fsinfo_attr) ;

fsinfo_attr : host | automount ;
```

7.4 *FSinfo* host definitions

A host declaration consists of three parts: a set of machine attribute data, a list of filesystems physically attached to the machine, and a list of additional statically mounted filesystems.

```
host        : "host" host_data list(filesystem) list(mount) ;
```

Each host must be declared in this way exactly once. Such things as the hardware address, the architecture and operating system types and the cluster name are all specified within the *host data*.

All the disks the machine has should then be described in the *list of filesystems*. When describing disks, you can specify what *volname* the disk/partition should have and all such entries are built up into a dictionary which can then be used for building the automounter maps.

The *list of mounts* specifies all the filesystems that should be statically mounted on the machine.

7.5 *FSinfo* host attributes

The host data, *host_data*, always includes the *hostname*. In addition, several other host attributes can be given.

```
host_data   : <hostname>
            | "{" list(host_attrs) "}" <hostname>
            ;

host_attrs  : host_attr "=" <string>
            | netif
            ;

host_attr   : "config"
            | "arch"
            | "os"
            | "cluster"
            ;
```

The *hostname* is, typically, the fully qualified hostname of the machine.

Examples:

```
host dylan.doc.ic.ac.uk

host {
    os = hpux
    arch = hp300
} dougal.doc.ic.ac.uk
```

The options that can be given as host attributes are shown below.

7.5.1 netif Option

This defines the set of network interfaces configured on the machine. The interface attributes collected by *FSinfo* are the IP address, subnet mask and hardware address. Multiple interfaces may be defined for hosts with several interfaces by an entry for each interface. The values given are sanity checked, but are currently unused for anything else.

```
netif       : "netif" <string> "{" list(netif_attrs) "}" ;

netif_attrs : netif_attr "=" <string> ;

netif_attr  : "inaddr" | "netmask" | "hwaddr" ;
```

Examples:

```
netif ie0 {
    inaddr  = 129.31.81.37
    netmask = 0xfffffe00
    hwaddr  = "08:00:20:01:a6:a5"
}

netif ec0 { }
```

7.5.2 config Option

This option allows you to specify configuration variables for the startup scripts (‘`rc`’ scripts). A simple string should immediately follow the keyword.

Example:

```
config "NFS_SERVER=true"
config "ZEPHYR=true"
```

This option is currently unsupported.

7.5.3 arch Option

This defines the architecture of the machine. For example:

```
arch = hp300
```

This is intended to be of use when building architecture specific mountmaps, however, the option is currently unsupported.

7.5.4 os Option

This defines the operating system type of the host. For example:

```
os = hpux
```

This information is used when creating the '`fstab`' files, for example in choosing which format to use for the '`fstab`' entries within the file.

7.5.5 cluster Option

This is used for specifying in which cluster the machine belongs. For example:

```
cluster = "theory"
```

The cluster is intended to be used when generating the automount maps, although it is currently unsupported.

7.6 *FSinfo* filesystems

The list of physically attached filesystems follows the machine attributes. These should define all the filesystems available from this machine, whether exported or not. In addition to the device name, filesystems have several attributes, such as filesystem type, mount options, and '`fsck`' pass number which are needed to generate '`fstab`' entries.

```
filesystem  : "fs" <device> "{" list(fs_data) "}" ;

fs_data     : fs_data_attr "=" <string>
            | mount
            ;

fs_data_attr
            : "fstype" | "opts" | "passno"
            | "freq" | "dumpset" | "log"
            ;
```

Here, `<`*device*`>` is the device name of the disk (for example, '`/dev/dsk/2s0`'). The device name is used for building the mount maps and for the '`fstab`' file. The attributes that can be specified are shown in the following section.

The *FSinfo* configuration file for `dylan.doc.ic.ac.uk` is listed below.

```
host dylan.doc.ic.ac.uk

fs /dev/dsk/0s0 {
fstype = swap
}

fs /dev/dsk/0s0 {
fstype = hfs
opts = rw,noquota,grpid
passno = 0;
freq = 1;
mount / { }
}

fs /dev/dsk/1s0 {
fstype = hfs
opts = defaults
passno = 1;
freq = 1;
mount /usr {
local {
```

```
exportfs "dougal eden dylan zebedee brian"
volname /nfs/hp300/local
}
}
}

fs /dev/dsk/2s0 {
fstype = hfs
opts = defaults
passno = 1;
freq = 1;
mount default {
exportfs "toytown_clients hangers_on"
volname /home/dylan/dk2
}
}

fs /dev/dsk/3s0 {
fstype = hfs
opts = defaults
passno = 1;
freq = 1;
mount default {
exportfs "toytown_clients hangers_on"
volname /home/dylan/dk3
}
}

fs /dev/dsk/5s0 {
fstype = hfs
opts = defaults
passno = 1;
freq = 1;
mount default {
exportfs "toytown_clients hangers_on"
volname /home/dylan/dk5
}
}
```

7.6.1 fstype Option

This specifies the type of filesystem being declared and will be placed into the 'fstab' file as is. The value of this option will be handed to mount as the filesystem type—it should have such values as 4.2, nfs or swap. The value is not examined for correctness.

There is one special case. If the filesystem type is specified as 'export' then the filesystem information will not be added to the host's '`fstab`' information, but it will still be visible on the network. This is useful for defining hosts which contain referenced volumes but which are not under full control of *FSinfo*.

Example:

```
fstype = swap
```

7.6.2 opts Option

This defines any options that should be given to **mount**(8) in the '`fstab`' file. For example:

```
opts = rw,nosuid,grpid
```

7.6.3 passno Option

This defines the **fsck**(8) pass number in which to check the filesystem. This value will be placed into the '`fstab`' file.

Example:

```
passno = 1
```

7.6.4 freq Option

This defines the interval (in days) between dumps. The value is placed as is into the '`fstab`' file.

Example:

```
freq = 3
```

7.6.5 mount Option

This defines the mountpoint at which to place the filesystem. If the mountpoint of the filesystem is specified as `default`, then the filesystem will be mounted in the automounter's tree under its volume name and the mount will automatically be inherited by the automounter.

Following the mountpoint, namespace information for the filesystem may be described. The options that can be given here are `exportfs`, `volname` and `sel`.

The format is:

```
mount         : "mount" vol_tree ;

vol_tree      : list(vol_tree_attr) ;

vol_tree_attr
              :  <string> "{" list(vol_tree_info) vol_tree "}" ;

vol_tree_info
              : "exportfs" <export-data>
              | "volname" <volname>
              | "sel" <selector-list>
              ;
```

Example:

```
mount default {
    exportfs "dylan dougal florence zebedee"
    volname /vol/andrew
}
```

In the above example, the filesystem currently being declared will have an entry placed into the '`exports`' file allowing the filesystem to be exported to the machines `dylan`, `dougal`, `florence` and `zebedee`. The volume name by which the filesystem will be referred to remotely, is '`/vol/andrew`'. By declaring the mountpoint to be `default`, the filesystem will be mounted on the local machine in the automounter tree, where *Amd* will automatically inherit the mount as '`/vol/andrew`'.

'`exportfs`'
: a string defining which machines the filesystem may be exported to. This is copied, as is, into the '`exports`' file—no sanity checking is performed on this string.

'`volname`'
: a string which declares the remote name by which to reference the filesystem. The string is entered into a dictionary and allows you to refer to this filesystem in other

places by this volume name.

'sel' a string which is placed into the automounter maps as a selector for the filesystem.

7.6.6 dumpset Option

This provides support for Imperial College's local file backup tools and is not documented further here.

7.6.7 log Option

Specifies the log device for the current filesystem. This is ignored if not required by the particular filesystem type.

7.7 *FSinfo* static mounts

Each host may also have a number of statically mounted filesystems. For example, the host may be a diskless workstation in which case it will have no `fs` declarations. In this case the `mount` declaration is used to determine from where its filesystems will be mounted. In addition to being added to the 'fstab' file, this information can also be used to generate a suitable 'bootparams' file.

```
mount          : "mount" <volname> list(localinfo) ;

localinfo      : localinfo_attr <string> ;

localinfo_attr
               : "as"
               | "from"
               | "fstype"
               | "opts"
               ;
```

The filesystem specified to be mounted will be searched for in the dictionary of volume names built when scanning the list of hosts' definitions.

The attributes have the following semantics:

'`from` *machine*'
: mount the filesystem from the machine with the hostname of *machine*.

'`as` *mountpoint*'
: mount the filesystem locally as the name given, in case this is different from the advertised volume name of the filesystem.

'`opts` *options*'
: native **mount**(8) options.

'`fstype` *type*'
: type of filesystem to be mounted.

An example:

```
mount /export/exec/hp300/local as /usr/local
```

If the mountpoint specified is either '`/`' or '`swap`', the machine will be considered to be booting off the net and this will be noted for use in generating a '`bootparams`' file for the host which owns the filesystems.

7.8 Defining an *Amd* Mount Map in *FSinfo*

The maps used by *Amd* can be constructed from *FSinfo* by defining all the automount trees. *FSinfo* takes all the definitions found and builds one map for each top level tree.

The automount tree is usually defined last. A single automount configuration will usually apply to an entire management domain. One `automount` declaration is needed for each *Amd* automount point. *FSinfo* determines whether the automount point is *direct* (see section 5.9 [Direct Automount Filesystem], page 37) or *indirect* (see section 5.12 [Top-level Filesystem], page 39). Direct automount points are distinguished by the fact that there is no underlying *automount_tree*.

```
automount    : "automount" opt(auto_opts) automount_tree ;

auto_opts    : "opts" <mount-options> ;

automount_tree
             : list(automount_attr)
             ;

automount_attr
             : <string> "=" <volname>
```

```
                  | <string> "->" <symlink>
                  | <string> "{" automount_tree "}"
                  ;
```

If <*mount-options*> is given, then it is the string to be placed in the maps for *Amd* for the `opts` option.

A *map* is typically a tree of filesystems, for example '`home`' normally contains a tree of filesystems representing other machines in the network.

A map can either be given as a name representing an already defined volume name, or it can be a tree. A tree is represented by placing braces after the name. For example, to define a tree '`/vol`', the following map would be defined:

```
automount /vol { }
```

Within a tree, the only items that can appear are more maps. For example:

```
automount /vol {
    andrew { }
    X11 { }
}
```

In this case, *FSinfo* will look for volumes named '`/vol/andrew`' and '`/vol/X11`' and a map entry will be generated for each. If the volumes are defined more than once, then *FSinfo* will generate a series of alternate entries for them in the maps.

Instead of a tree, either a link (*name* `->` *destination*) or a reference can be specified (*name* `=` *destination*). A link creates a symbolic link to the string specified, without further processing the entry. A reference will examine the destination filesystem and optimise the reference. For example, to create an entry for `njw` in the '`/homes`' map, either of the two forms can be used:

```
automount /homes {
    njw -> /home/dylan/njw
}
```

or

```
automount /homes {
    njw = /home/dylan/njw
```

```
}
```

In the first example, when '`/homes/njw`' is referenced from *Amd*, a link will be created leading to '`/home/dylan/njw`' and the automounter will be referenced a second time to resolve this filename. The map entry would be:

```
njw type:=link;fs:=/home/dylan/njw
```

In the second example, the destination directory is analysed and found to be in the filesystem '`/home/dylan`' which has previously been defined in the maps. Hence the map entry will look like:

```
njw rhost:=dylan;rfs:=/home/dylan;sublink:=njw
```

Creating only one symbolic link, and one access to *Amd*.

7.9 *FSinfo* Command Line Options

FSinfo is started from the command line by using the command:

```
fsinfo [options] files ...
```

The input to *FSinfo* is a single set of definitions of machines and automount maps. If multiple files are given on the command-line, then the files are concatenated together to form the input source. The files are passed individually through the C pre-processor before being parsed.

Several options define a prefix for the name of an output file. If the prefix is not specified no output of that type is produced. The suffix used will correspond either to the hostname to which a file belongs, or to the type of output if only one file is produced. Dumpsets and the '`bootparams`' file are in the latter class. To put the output into a subdirectory simply put a '`/`' at the end of the prefix, making sure that the directory has already been made before running '`fsinfo`'.

7.9.1 `-a` *autodir*

Specifies the directory name in which to place the automounter's mountpoints. This defaults to '`/a`'. Some sites have the autodir set to be '`/amd`', and this would be achieved by:

```
fsinfo -a /amd ...
```

7.9.2 `-b` *bootparams*

This specifies the prefix for the '`bootparams`' filename. If it is not given, then the file will not be generated. The '`bootparams`' file will be constructed for the destination machine and will be placed into a file named '`bootparams`' and prefixed by this string. The file generated contains a list of entries describing each diskless client that can boot from the destination machine.

As an example, to create a '`bootparams`' file in the directory '`generic`', the following would be used:

```
fsinfo -b generic/ ...
```

7.9.3 `-d` *dumpsets*

This specifies the prefix for the '`dumpsets`' file. If it is not specified, then the file will not be generated. The file will be for the destination machine and will be placed into a filename '`dumpsets`', prefixed by this string. The '`dumpsets`' file is for use by Imperial College's local backup system.

For example, to create a dumpsets file in the directory '`generic`', then you would use the following:

```
fsinfo -d generic/ ...
```

7.9.4 `-e` *exportfs*

Defines the prefix for the '`exports`' files. If it is not given, then the file will not be generated. For each machine defined in the configuration files as having disks, an '`exports`' file is constructed and given a filename determined by the name of the machine, prefixed with this string. If a machine is defined as diskless, then no '`exports`' file will be created for it. The files contain entries for directories on the machine that may be exported to clients.

Example: To create the '`exports`' files for each diskful machine and place them into the directory '`exports`':

```
fsinfo -e exports/ ...
```

7.9.5 `-f` *fstab*

This defines the prefix for the '`fstab`' files. The files will only be created if this prefix is defined. For each machine defined in the configuration files, a '`fstab`' file is created with the filename determined by prefixing this string with the name of the machine. These files contain entries for filesystems and partitions to mount at boot time.

Example, to create the files in the directory '`fstabs`':

```
fsinfo -f fstabs/ ...
```

7.9.6 `-h` *hostname*

Defines the hostname of the destination machine to process for. If this is not specified, it defaults to the local machine name, as returned by **gethostname**(2).

Example:

```
fsinfo -h dylan.doc.ic.ac.uk ...
```

7.9.7 `-m` *mount-maps*

Defines the prefix for the automounter files. The maps will only be produced if this prefix is defined. The mount maps suitable for the network defined by the configuration files will be placed into files with names calculated by prefixing this string to the name of each map.

For example, to create the automounter maps and place them in the directory '`automaps`':

```
fsinfo -m automaps/ ...
```

7.9.8 `-q`

Selects quiet mode. *FSinfo* suppress the "running commentary" and only outputs any error messages which are generated.

7.9.9 `-v`

Selects verbose mode. When this is activated, the program will display more messages, and display all the information discovered when performing the semantic analysis phase. Each verbose message is output to '`stdout`' on a line starting with a '`#`' character.

7.9.10 `-D` *name[=defn]*

Defines a symbol *name* for the preprocessor when reading the configuration files. Equivalent to `#define` directive.

7.9.11 `-I` *directory*

This option is passed into the preprocessor for the configuration files. It specifies directories in which to find include files

7.9.12 `-U` *name*

Removes any initial definition of the symbol *name*. Inverse of the `-D` option.

7.10 Errors produced by *FSinfo*

The following table documents the errors and warnings which *FSinfo* may produce.

`can't open` *filename* `for writing`
: Occurs if any errors are encountered when opening an output file.

`unknown host attribute`
: Occurs if an unrecognised keyword is used when defining a host.

`unknown filesystem attribute`
: Occurs if an unrecognised keyword is used when defining a host's filesystems.

`not allowed '/' in a directory name`
: When reading the configuration input, if there is a filesystem definition which contains a pathname with multiple directories for any part of the mountpoint element, and it is not a single absolute path, then this message will be produced by the parser.

`unknown directory attribute`
: If an unknown keyword is found while reading the definition of a hosts's filesystem mount option.

`unknown mount attribute`
: Occurs if an unrecognised keyword is found while parsing the list of static mounts.

`" expected`
: Occurs if an unescaped newline is found in a quoted string.

`unknown \ sequence`
: Occurs if an unknown escape sequence is found inside a string. Within a string, you can give the standard C escape sequences for strings, such as newlines and tab characters.

filename: `cannot open for reading`
: If a file specified on the command line as containing configuration data could not be opened.

`end of file within comment`
: A comment was unterminated before the end of one of the configuration files.

`host field "`*field-name*`" already set`
: If duplicate definitions are given for any of the fields with a host definition.

`duplicate host` *hostname*`!`
: If a host has more than one definition.

`netif field` *field-name* `already set`
: Occurs if you attempt to define an attribute of an interface more than once.

`malformed IP dotted quad:` *address*
: If the Internet address of an interface is incorrectly specified. An Internet address definition is handled to **inet_addr**(3N) to see if it can cope. If not, then this message will be displayed.

`malformed netmask:` *netmask*
: If the netmask cannot be decoded as though it were a hexadecimal number, then this message will be displayed. It will typically be caused by incorrect characters in the *netmask* value.

`fs field "`*field-name*`" already set`
: Occurs when multiple definitions are given for one of the attributes of a host's filesystem.

`mount tree field "`*field-name*`" already set`
: Occurs when the *field-name* is defined more than once during the definition of a filesystems mountpoint.

`mount field "`*field-name*`" already set`
: Occurs when a static mount has multiple definitions of the same field.

`no disk mounts on` *hostname*
: If there are no static mounts, nor local disk mounts specified for a machine, this message will be displayed.

host`:`*device* `needs field "`*field-name*`"`
: Occurs when a filesystem is missing a required field. *field-name* could be one of `fstype`, `opts`, `passno` or `mount`.

filesystem `has a volname but no exportfs data`
: Occurs when a volume name is declared for a file system, but the string specifying what machines the filesystem can be exported to is missing.

`sub-directory` *directory* `of` *directory-tree* `starts with '/'`
: Within the filesystem specification for a host, if an element *directory* of the mountpoint begins with a '`/`' and it is not the start of the tree.

host`:`*device* `has no mount point`
: Occurs if the '`mount`' option is not specified for a host's filesystem.

host`:`*device* `has more than one mount point`
: Occurs if the mount option for a host's filesystem specifies multiple trees at which to place the mountpoint.

`no volname given for` *host*`:`*device*
: Occurs when a filesystem is defined to be mounted on '`default`', but no volume name is given for the file system, then the mountpoint cannot be determined.

host`:mount field specified for swap partition`
: Occurs if a mountpoint is given for a filesystem whose type is declared to be `swap`.

`ambiguous mount:` *volume* `is a replicated filesystem`
: If several filesystems are declared as having the same volume name, they will be considered replicated filesystems. To mount a replicated filesystem statically, a specific host will need to be named, to say which particular copy to try and mount, else this error will result.

`cannot determine localname since volname` *volume* `is not uniquely defined`
: If a volume is replicated and an attempt is made to mount the filesystem statically without specifying a local mountpoint, *FSinfo* cannot calculate a mountpoint, as the desired pathname would be ambiguous.

`volname` *volume* `is unknown`
: Occurs if an attempt is made to mount or reference a volume name which has not been declared during the host filesystem definitions.

`volname` *volume* `not exported from` *machine*
: Occurs if you attempt to mount the volume *volume* from a machine which has not declared itself to have such a filesystem available.

`network booting requires both root and swap areas`
: Occurs if a machine has mount declarations for either the root partition or the swap area, but not both. You cannot define a machine to only partially boot via the network.

`unknown volname` *volume* `automounted` *[* `on <name>` *]*
: Occurs if *volume* is used in a definition of an automount map but the volume name has not been declared during the host filesystem definitions.

`not allowed '/' in a directory name`
: Occurs when a pathname with multiple directory elements is specified as the name for an automounter tree. A tree should only have one name at each level.

device `has duplicate exportfs data`
: Produced if the '`exportfs`' option is used multiple times within the same branch of a filesytem definition. For example, if you attempt to set the '`exportfs`' data at different levels of the mountpoint directory tree.

`sub-directory of` *directory-tree* `is named "default"`
: '`default`' is a keyword used to specify if a mountpoint should be automatically calculated by *FSinfo*. If you attempt to specify a directory name as this, it will use the filename of '`default`' but will produce this warning.

`pass number for` *host*`:`*device* `is non-zero`
: Occurs if *device* has its '`fstype`' declared to be '`swap`' or '`export`' and the **fsck**(8) pass number is set. Swap devices should not be fsck'd. See section 7.6.1 [FSinfo filesystems fstype], page 52

`dump frequency for` *host*`:`*device* `is non-zero`
: Occurs if *device* has its '`fstype`' declared to be '`swap`' or '`export`' and the '`dump`' option is set to a value greater than zero. Swap devices should not be dumped.

8. Examples

8.1 User Filesystems

With more than one fileserver, the directories most frequently cross-mounted are those containing user home directories. A common convention used at Imperial College is to mount the user disks under `/home/`*machine*.

Typically, the '`/etc/fstab`' file contained a long list of entries such as:

machine`:/home/`*machine* `/home/`*machine* `nfs ...`

for each fileserver on the network.

There are numerous problems with this system. The mount list can become quite large and some of the machines may be down when a system is booted. When a new fileserver is installed, '`/etc/fstab`' must be updated on every machine, the mount directory created and the filesystem mounted.

In many environments most people use the same few workstations, but it is convenient to go to a colleague's machine and access your own files. When a server goes down, it can cause a process on a client machine to hang. By minimising the mounted filesystems to only include those actively being used, there is less chance that a filesystem will be mounted when a server goes down.

The following is a short extract from a map taken from a research fileserver at Imperial College.

Note the entry for '`localhost`' which is used for users such as the operator ('`opr`') who have a home directory on most machine as '`/home/localhost/opr`'.

```
/defaults       opts:=rw,intr,grpid,nosuid
charm           host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/xd0g
#
...

#
localhost       type:=link;fs:=${host}
...
```

```
#
# dylan has two user disks so have a
# top directory in which to mount them.
#
dylan           type:=auto;fs:=${map};pref:=${key}/
#
dylan/dk2       host!=dylan;type:=nfs;rhost:=dylan;rfs:=/home/${key} \
                host==dylan;type:=ufs;dev:=/dev/dsk/2s0
#
dylan/dk5       host!=dylan;type:=nfs;rhost:=dylan;rfs:=/home/${key} \
                host==dylan;type:=ufs;dev:=/dev/dsk/5s0
...
#
toytown         host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/xy1g
...
#
zebedee         host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/dsk/1s0
#
# Just for access...
#
gould           type:=auto;fs:=${map};pref:=${key}/
gould/staff     host!=gould;type:=nfs;rhost:=gould;rfs:=/home/${key}
#
gummo           host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key}
...
```

This map is shared by most of the machines listed so on those systems any of the user disks is accessible via a consistent name. *Amd* is started with the following command

```
amd /home amd.home
```

Note that when mounting a remote filesystem, the *automounted* mount point is referenced, so that the filesystem will be mounted if it is not yet (at the time the remote 'mountd' obtains the file handle).

8.2 Home Directories

One convention for home directories is to locate them in '/homes' so user 'jsp''s home directory is '/homes/jsp'. With more than a single fileserver it is convenient to spread user files across several machines. All that is required is a mount-map which converts login names to an automounted directory.

Such a map might be started by the command:

```
amd /homes amd.homes
```

where the map '`amd.homes`' contained the entries:

```
/defaults   type:=link   # All the entries are of type:=link
jsp         fs:=/home/charm/jsp
njw         fs:=/home/dylan/dk5/njw
...
phjk        fs:=/home/toytown/ai/phjk
sjv         fs:=/home/ganymede/sjv
```

Whenever a login name is accessed in '`/homes`' a symbolic link appears pointing to the real location of that user's home directory. In this example, '`/homes/jsp`' would appear to be a symbolic link pointing to '`/home/charm/jsp`'. Of course, '`/home`' would also be an automount point.

This system causes an extra level of symbolic links to be used. Although that turns out to be relatively inexpensive, an alternative is to directly mount the required filesystems in the '`/homes`' map. The required map is simple, but long, and its creation is best automated. The entry for '`jsp`' could be:

```
jsp   -sublink:=${key};rfs:=/home/charm \
               host==charm;type:=ufs;dev:=/dev/xd0g \
               host!=charm;type:=nfs;rhost:=charm
```

This map can become quite big if it contains a large number of entries. By combining two other features of *Amd* it can be greatly simplified.

First the UFS partitions should be mounted under the control of '`/etc/fstab`', taking care that they are mounted in the same place that *Amd* would have automounted them. In most cases this would be something like '`/a/`*host*`/home/`*host*' and '`/etc/fstab`' on host '`charm`' would have a line:

```
/dev/xy0g /a/charm/home/charm 4.2 rw,nosuid,grpid 1 5
```

The map can then be changed to:

```
/defaults    type:=nfs;sublink:=${key};opts:=rw,intr,nosuid,grpid
jsp          rhost:=charm;rfs:=/home/charm
njw          rhost:=dylan;rfs:=/home/dylan/dk5
```

```
...
phjk         rhost:=toytown;rfs:=/home/toytown;sublink:=ai/${key}
sjv          rhost:=ganymede;rfs:=/home/ganymede
```

This map operates as usual on a remote machine (*ie* `${host}` not equal to `${rhost}`). On the machine where the filesystem is stored (*ie* `${host}` equal to `${rhost}`), *Amd* will construct a local filesystem mount point which corresponds to the name of the locally mounted UFS partition. If *Amd* is started with the "-r" option then instead of attempting an NFS mount, *Amd* will simply inherit the UFS mount (see section 5.14 [Inheritance Filesystem], page 39). If "-r" is not used then a loopback NFS mount will be made. This type of mount is known to cause a deadlock on many systems.

8.3 Architecture Sharing

Often a filesystem will be shared by machines of different architectures. Separate trees can be maintained for the executable images for each architecture, but it may be more convenient to have a shared tree, with distinct subdirectories.

A shared tree might have the following structure on the fileserver (called '`fserver`' in the example):

```
local/tex
local/tex/fonts
local/tex/lib
local/tex/bin
local/tex/bin/sun3
local/tex/bin/sun4
local/tex/bin/hp9000
...
```

In this example, the subdirectories of '`local/tex/bin`' should be hidden when accessed via the automount point (conventionally '`/vol`'). A mount-map for '`/vol`' to achieve this would look like:

```
/defaults   sublink:=${/key};rhost:=fserver;type:=link
tex         type:=auto;fs:=${map};pref:=${key}/
tex/fonts   host!=fserver;type:=nfs;rfs:=/vol/tex \
            host==fserver;fs:=/usr/local/tex
tex/lib     host!=fserver;type:=nfs;rfs:=/vol/tex \
            host==fserver;fs:=/usr/local/tex
tex/bin     -sublink:=${/key}/${arch} host!=fserver;type:=nfs;rfs:=/vol/tex \
            host:=fserver;fs:=/usr/local/tex
```

When '/vol/tex/bin' is referenced, the current machine architecture is automatically appended to the path by the ${sublink} variable. This means that users can have '/vol/tex/bin' in their 'PATH' without concern for architecture dependencies.

8.4 Wildcard names & Replicated Servers

By using the wildcard facility, *Amd* can *overlay* an existing directory with additional entries. The system files are usually mounted under '/usr'. If instead *Amd* is mounted on '/usr', additional names can be overlayed to augment or replace names in the "master" '/usr'. A map to do this would have the form:

```
local  type:=auto;fs:=local-map
share  type:=auto;fs:=share-map
*      -type:=nfs;rfs:=/export/exec/${arch};sublink:="${key}" \
        rhost:=fserv1  rhost:=fserv2  rhost:=fserv3
```

Note that the assignment to ${sublink} is surrounded by double quotes to prevent the incoming key from causing the map to be misinterpreted. This map has the effect of directing any access to '/usr/local' or '/usr/share' to another automount point.

In this example, it is assumed that the '/usr' files are replicated on three fileservers: 'fserv1', 'fserv2' and 'fserv3'. For any references other than to 'local' and 'share' one of the servers is used and a symbolic link to ${autodir}/${rhost}/export/exec/${arch}/*whatever* is returned once an appropriate filesystem has been mounted.

8.5 'rwho' servers

The '/usr/spool/rwho' directory is a good candidate for automounting. For efficiency reasons it is best to capture the rwho data on a small number of machines and then mount that information onto a large number of clients. The data written into the rwho files is byte order dependent so only servers with the correct byte ordering can be used by a client:

```
/defaults         type:=nfs
usr/spool/rwho    -byte==little;rfs:=/usr/spool/rwho \
                      rhost:=vaxA  rhost:=vaxB \
                  || -rfs:=/usr/spool/rwho \
                      rhost:=sun4  rhost:=hp300
```

8.6 '/vol'

'/vol' is used as a catch-all for volumes which do not have other conventional names.

Below is part of the '/vol' map for the domain 'doc.ic.ac.uk'. The 'r+d' tree is used for new or experimental software that needs to be available everywhere without installing it on all the fileservers. Users wishing to try out the new software then simply include '/vol/r+d/{bin,ucb}' in their path.

The main tree resides on one host 'gould.doc.ic.ac.uk', which has different 'bin', 'etc', 'lib' and 'ucb' sub-directories for each machine architecture. For example, '/vol/r+d/bin' for a Sun-4 would be stored in the sub-directory 'bin/sun4' of the filesystem '/usr/r+d'. When it was accessed a symbolic link pointing to '/a/gould/usr/r+d/bin/sun4' would be returned.

```
/defaults    type:=nfs;opts:=rw,grpid,nosuid,intr,soft
wp           -opts:=rw,grpid,nosuid;rhost:=charm \
             host==charm;type:=link;fs:=/usr/local/wp \
             host!=charm;type:=nfs;rfs:=/vol/wp
...
#
src          -opts:=rw,grpid,nosuid;rhost:=charm \
             host==charm;type:=link;fs:=/usr/src \
             host!=charm;type:=nfs;rfs:=/vol/src
#
r+d          type:=auto;fs:=${map};pref:=r+d/
# per architecture bin,etc,lib&ucb...
r+d/bin      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/etc      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/include  rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/lib      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/man      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/src      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/ucb      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
# hades pictures
pictures     -opts:=rw,grpid,nosuid;rhost:=thpfs \
             host==thpfs;type:=link;fs:=/nbsd/pictures \
             host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=pictures
# hades tools
hades        -opts:=rw,grpid,nosuid;rhost:=thpfs \
             host==thpfs;type:=link;fs:=/nbsd/hades \
             host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=hades
# bsd tools for hp.
bsd          -opts:=rw,grpid,nosuid;arch==hp9000;rhost:=thpfs \
             host==thpfs;type:=link;fs:=/nbsd/bsd \
             host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=bsd
```

9. Internals

9.1 Log Messages

In the following sections a brief explanation is given of some of the log messages made by *Amd*. Where the message is in '`typewriter`' font, it corresponds exactly to the message produced by *Amd*. Words in *italic* are replaced by an appropriate string. Variables, `${var}`, indicate that the value of the appropriate variable is output.

Log messages are either sent direct to a file, or logged via the **syslog**(3) mechanism. Messages are logged with facility '`LOG_DAEMON`' when using **syslog**(3). In either case, entries in the file are of the form:

date-string *hostname* `amd[`*pid*`]` *message*

9.1.1 Fatal errors

Amd attempts to deal with unusual events. Whenever it is not possible to deal with such an error, *Amd* will log an appropriate message and, if it cannot possibly continue, will either exit or abort. These messages are selected by '`-x fatal`' on the command line. When **syslog**(3) is being used, they are logged with level '`LOG_FATAL`'. Even if *Amd* continues to operate it is likely to remain in a precarious state and should be restarted at the earliest opportunity.

`Attempting to inherit not-a-filesystem`
: The prototype mount point created during a filesystem restart did not contain a reference to the restarted filesystem. This erorr "should never happen".

`Can't bind to domain "`*NIS-domain*`"`
: A specific NIS domain was requested on the command line, but no server for that domain is available on the local net.

`Can't determine IP address of this host (`*hostname*`)`
: When *Amd* starts it determines its own IP address. If this lookup fails then *Amd* cannot continue. The hostname it looks up is that obtained returned by **gethostname**(2) system call.

`Can't find root file handle for` *automount point*
: *Amd* creates its own file handles for the automount points. When it mounts itself as

a server, it must pass these file handles to the local kernel. If the filehandle is not obtainable the mount point is ignored. This error "should never happen".

`Must be root to mount filesystems (euid =` *euid*`)`
: To prevent embarrassment, *Amd* makes sure it has appropriate system privileges. This amounts to having an euid of 0. The check is made after argument processing complete to give non-root users a chance to access the "-v" option.

`No work to do - quitting`
: No automount points were given on the command line and so there is no work to do.

`Out of memory in realloc`
: While attempting to realloc some memory, the memory space available to *Amd* was exhausted. This is an unrecoverable error.

`Out of memory`
: While attempting to malloc some memory, the memory space available to *Amd* was exhausted. This is an unrecoverable error.

`cannot create rpc/udp service`
: Either the NFS or AMQ endpoint could not be created.

`gethostname:` *description*
: The **gethostname**(2) system call failed during startup.

`host name is not set`
: The **gethostname**(2) system call returned a zero length host name. This can happen if *Amd* is started in single user mode just after booting the system.

`ifs_match called!`
: An internal error occurred while restarting a pre-mounted filesystem. This error "should never happen".

`mount_afs:` *description*
: An error occured while *Amd* was mounting itself.

`run_rpc failed`
: Somehow the main NFS server loop failed. This error "should never happen".

`unable to free rpc arguments in amqprog_1`
: The incoming arguments to the AMQ server could not be free'ed.

`unable to free rpc arguments in nfs_program_1`
: The incoming arguments to the NFS server could not be free'ed.

`unable to register (AMQ_PROGRAM, AMQ_VERSION, udp)`
: The AMQ server could not be registered with the local portmapper or the internal RPC dispatcher.

`unable to register (NFS_PROGRAM, NFS_VERSION, 0)`
: The NFS server could not be registered with the internal RPC dispatcher.

9.1.2 Info messages

Amd generates information messages to record state changes. These messages are selected by '`-x info`' on the command line. When **syslog**(3) is being used, they are logged with level '`LOG_INFO`'.

The messages listed below can be generated and are in a format suitable for simple statistical analysis. *mount-info* is the string that is displayed by *Amq* in its mount information column and placed in the system mount table.

`mount of "${`*path*`}" on ${`*fs*`} timed out`
: Attempts to mount a filesystem for the given automount point have failed to complete within 30 seconds.

`"${`*path*`}" forcibly timed out`
: An automount point has been timed out by the *Amq* command.

`restarting` *mount-info* `on ${`*fs*`}`
: A pre-mounted file system has been noted.

`"${`*path*`}" has timed out`
: No access to the automount point has been made within the timeout period.

`file server ${`*rhost*`} is down - timeout of "${`*path*`}" ignored`
: An automount point has timed out, but the corresponding file server is known to be down. This message is only produced once for each mount point for which the server is down.

`Re-synchronizing cache for map ${`*map*`}`
: The named map has been modified and the internal cache is being re-synchronized.

`Filehandle denied for "${`*rhost*`}:${`*rfs*`}"`
: The mount daemon refused to return a file handle for the requested filesystem.

`Filehandle error for "${`*rhost*`}:${`*rfs*`}":` *description*
: The mount daemon gave some other error for the requested filesystem.

`file server ${`*rhost*`} type nfs starts up`
: A new NFS file server has been referenced and is known to be up.

`file server ${`*rhost*`} type nfs starts down`
: A new NFS file server has been referenced and is known to be down.

`file server ${`*rhost*`} type nfs is up`
: An NFS file server that was previously down is now up.

`file server ${`*rhost*`} type nfs is down`
: An NFS file server that was previously up is now down.

`Finishing with status` *exit-status*
: *Amd* is about to exit with the given exit status.

mount-info `mounted fstype ${`*type*`} on ${`*fs*`}`
: A new file system has been mounted.

mount-info `restarted fstype ${`*type*`} on ${`*fs*`}`
: *Amd* is using a pre-mounted filesystem to satisfy a mount request.

mount-info `unmounted fstype ${`*type*`} from ${`*fs*`}`
: A file system has been unmounted.

mount-info `unmounted fstype ${`*type*`} from ${`*fs*`} link ${`*fs*`}/${`*sublink*`}`
: A file system of which only a sub-directory was in use has been unmounted.

Acknowledgements & Trademarks

Thanks to the Formal Methods Group at Imperial College for suffering patiently while *Amd* was being developed on their machines.

Thanks to the many people who have helped with the development of *Amd*, especially Piete Brooks at the Cambridge University Computing Lab for many hours of testing, experimentation and discussion.

Table of Contents