NAME

intro — introduction to file formats

DESCRIPTION

This section contains documentation on binary and configuration file formats.

HISTORY

intro appeared in NetBSD 1.4.

NAME

a.out — format of executable binary files

SYNOPSIS

```
#include <sys/types.h>
#include <a.out.h>
```

DESCRIPTION

The include file ⟨a.out.h⟩ declares three structures and several macros. The structures describe the format of executable machine code files (‘binaries’) on the system.

A binary file consists of up to 7 sections. In order, these sections are:

exec header	Contains parameters used by the kernel to load a binary file into memory and execute it, and by the link editor ld(1) to combine a binary file with other binary files. This section is the only mandatory one.
text segment	Contains machine code and related data that are loaded into memory when a program executes. May be loaded read-only.
data segment	Contains initialized data; always loaded into writable memory.
text relocations	Contains records used by the link editor to update pointers in the text segment when combining binary files.
data relocations	Like the text relocation section, but for data segment pointers.
symbol table	Contains records used by the link editor to cross reference the addresses of named variables and functions (‘symbols’) between binary files.
string table	Contains the character strings corresponding to the symbol names.

Every binary file begins with an *exec* structure:

```
struct exec {
        unsigned long  a_midmag;
        unsigned long  a_text;
        unsigned long  a_data;
        unsigned long  a_bss;
        unsigned long  a_syms;
        unsigned long  a_entry;
        unsigned long  a_trsize;
        unsigned long  a_drsize;
};
```

The fields have the following functions:

a_midmag This field is stored in network byte-order so that binaries for machines with alternative byte orders can be distinguished. It has a number of sub-components accessed by the macros N_GETFLAG(), N_GETMID(), and N_GETMAGIC(), and set by the macro N_SETMAGIC().

The macro N_GETFLAG() returns a few flags:

EX_DYNAMIC indicates that the executable requires the services of the run-time link editor.

EX_PIC indicates that the object contains position independent code. This flag is set by as(1) when given the ‘-k’ flag and is preserved by ld(1) if necessary.

If both EX_DYNAMIC and EX_PIC are set, the object file is a position independent executable image (e.g. a shared library), which is to be loaded into the process address space by the run-time link editor.

The macro `N_GETMID()` returns the machine-id. This indicates which machine(s) the binary is intended to run on.

`N_GETMAGIC()` specifies the magic number, which uniquely identifies binary files and distinguishes different loading conventions. The field must contain one of the following values:

`OMAGIC` The text and data segments immediately follow the header and are contiguous. The kernel loads both text and data segments into writable memory.

`NMAGIC` As with `OMAGIC`, text and data segments immediately follow the header and are contiguous. However, the kernel loads the text into read-only memory and loads the data into writable memory at the next page boundary after the text.

`ZMAGIC` The kernel loads individual pages on demand from the binary. The header, text segment and data segment are all padded by the link editor to a multiple of the page size. Pages that the kernel loads from the text segment are read-only, while pages from the data segment are writable.

a_text Contains the size of the text segment in bytes.

a_data Contains the size of the data segment in bytes.

a_bss Contains the number of bytes in the 'bss segment' and is used by the kernel to set the initial break (`brk`(2)) after the data segment. The kernel loads the program so that this amount of writable memory appears to follow the data segment and initially reads as zeroes.

a_syms Contains the size in bytes of the symbol table section.

a_entry Contains the address in memory of the entry point of the program after the kernel has loaded it; the kernel starts the execution of the program from the machine instruction at this address.

a_trsize Contains the size in bytes of the text relocation table.

a_drsize Contains the size in bytes of the data relocation table.

The `a.out.h` include file defines several macros which use an *exec* structure to test consistency or to locate section offsets in the binary file.

N_BADMAG(*exec*) Nonzero if the *a_magic* field does not contain a recognized value.

N_TXTOFF(*exec*) The byte offset in the binary file of the beginning of the text segment.

N_SYMOFF(*exec*) The byte offset of the beginning of the symbol table.

N_STROFF(*exec*) The byte offset of the beginning of the string table.

Relocation records have a standard format which is described by the *relocation_info* structure:

```
struct relocation_info {
        int             r_address;
        unsigned int    r_symbolnum : 24,
                        r_pcrel : 1,
                        r_length : 2,
                        r_extern : 1,
                        r_baserel : 1,
                        r_jmptable : 1,
                        r_relative : 1,
```

```
                        r_copy : 1;
};
```

The *relocation_info* fields are used as follows:

r_address Contains the byte offset of a pointer that needs to be link-edited. Text relocation offsets are reckoned from the start of the text segment, and data relocation offsets from the start of the data segment. The link editor adds the value that is already stored at this offset into the new value that it computes using this relocation record.

r_symbolnum Contains the ordinal number of a symbol structure in the symbol table (it is *not* a byte offset). After the link editor resolves the absolute address for this symbol, it adds that address to the pointer that is undergoing relocation. (If the *r_extern* bit is clear, the situation is different; see below.)

r_pcrel If this is set, the link editor assumes that it is updating a pointer that is part of a machine code instruction using pc-relative addressing. The address of the relocated pointer is implicitly added to its value when the running program uses it.

r_length Contains the log base 2 of the length of the pointer in bytes; 0 for 1-byte displacements, 1 for 2-byte displacements, 2 for 4-byte displacements.

r_extern Set if this relocation requires an external reference; the link editor must use a symbol address to update the pointer. When the *r_extern* bit is clear, the relocation is 'local'; the link editor updates the pointer to reflect changes in the load addresses of the various segments, rather than changes in the value of a symbol (except when *r_baserel* is also set, see below). In this case, the content of the *r_symbolnum* field is an *n_type* value (see below); this type field tells the link editor what segment the relocated pointer points into.

r_baserel If set, the symbol, as identified by the *r_symbolnum* field, is to be relocated to an offset into the Global Offset Table. At run-time, the entry in the Global Offset Table at this offset is set to be the address of the symbol.

r_jmptable If set, the symbol, as identified by the *r_symbolnum* field, is to be relocated to an offset into the Procedure Linkage Table.

r_relative If set, this relocation is relative to the (run-time) load address of the image this object file is going to be a part of. This type of relocation only occurs in shared objects.

r_copy If set, this relocation record identifies a symbol whose contents should be copied to the location given in *r_address*. The copying is done by the run-time link-editor from a suitable data item in a shared object.

Symbols map names to addresses (or more generally, strings to values). Since the link-editor adjusts addresses, a symbol's name must be used to stand for its address until an absolute value has been assigned. Symbols consist of a fixed-length record in the symbol table and a variable-length name in the string table. The symbol table is an array of *nlist* structures:

```
struct nlist {
        union {
                char    *n_name;
                long    n_strx;
        } n_un;
        unsigned char  n_type;
        char           n_other;
        short          n_desc;
        unsigned long  n_value;
```

```
};
```

The fields are used as follows:

n_un.n_strx Contains a byte offset into the string table for the name of this symbol. When a program accesses a symbol table with the nlist(3) function, this field is replaced with the *n_un.n_name* field, which is a pointer to the string in memory.

n_type Used by the link editor to determine how to update the symbol's value. The *n_type* field is broken down into three sub-fields using bitmasks. The link editor treats symbols with the N_EXT type bit set as 'external' symbols and permits references to them from other binary files. The N_TYPE mask selects bits of interest to the link editor:

- N_UNDF An undefined symbol. The link editor must locate an external symbol with the same name in another binary file to determine the absolute value of this symbol. As a special case, if the *n_value* field is nonzero and no binary file in the link-edit defines this symbol, the link-editor will resolve this symbol to an address in the bss segment, reserving an amount of bytes equal to *n_value*. If this symbol is undefined in more than one binary file and the binary files do not agree on the size, the link editor chooses the greatest size found across all binaries.
- N_ABS An absolute symbol. The link editor does not update an absolute symbol.
- N_TEXT A text symbol. This symbol's value is a text address and the link editor will update it when it merges binary files.
- N_DATA A data symbol; similar to N_TEXT but for data addresses. The values for text and data symbols are not file offsets but addresses; to recover the file offsets, it is necessary to identify the loaded address of the beginning of the corresponding section and subtract it, then add the offset of the section.
- N_BSS A bss symbol; like text or data symbols but has no corresponding offset in the binary file.
- N_FN A filename symbol. The link editor inserts this symbol before the other symbols from a binary file when merging binary files. The name of the symbol is the filename given to the link editor, and its value is the first text address from that binary file. Filename symbols are not needed for link-editing or loading, but are useful for debuggers.

The N_STAB mask selects bits of interest to symbolic debuggers such as gdb(1); the values are described in stab(5).

n_other This field provides information on the nature of the symbol independent of the symbol's location in terms of segments as determined by the *n_type* field. Currently, the lower 4 bits of the *n_other* field hold one of two values: AUX_FUNC and AUX_OBJECT (see ⟨link.h⟩ for their definitions). AUX_FUNC associates the symbol with a callable function, while AUX_OBJECT associates the symbol with data, irrespective of their locations in either the text or the data segment. This field is intended to be used by ld(1) for the construction of dynamic executables.

n_desc Reserved for use by debuggers; passed untouched by the link editor. Different debuggers use this field for different purposes.

n_value Contains the value of the symbol. For text, data and bss symbols, this is an address; for other symbols (such as debugger symbols), the value may be arbitrary.

The string table consists of an *unsigned long* length followed by null-terminated symbol strings. The length represents the size of the entire table in bytes, so its minimum value (or the offset of the first string) is always 4 on 32-bit machines.

SEE ALSO

`as`(1), `gdb`(1), `ld`(1), `brk`(2), `execve`(2), `nlist`(3), `core`(5), `elf`(5), `link`(5), `stab`(5)

HISTORY

The `a.out.h` include file appeared in Version 7 AT&T UNIX.

BUGS

Nobody seems to agree on what *bss* stands for.

New binary file formats may be supported in the future, and they probably will not be compatible at any level with this ancient format.

NAME

access − Postfix SMTP server access table

SYNOPSIS

postmap /etc/postfix/access

postmap -q "*string***" /etc/postfix/access**

postmap -q - /etc/postfix/access <*inputfile*

DESCRIPTION

This document describes access control on remote SMTP client information: host names, network addresses, and envelope sender or recipient addresses; it is implemented by the Postfix SMTP server. See **header_checks**(5) or **body_checks**(5) for access control on the content of email messages.

Normally, the **access**(5) table is specified as a text file that serves as input to the **postmap**(1) command. The result, an indexed file in **dbm** or **db** format, is used for fast searching by the mail system. Execute the command "**postmap /etc/postfix/access**" to rebuild an indexed file after changing the corresponding text file.

When the table is provided via other means such as NIS, LDAP or SQL, the same lookups are done as for ordinary indexed files.

Alternatively, the table can be provided as a regular-expression map where patterns are given as regular expressions, or lookups can be directed to TCP-based server. In those cases, the lookups are done in a slightly different way as described below under "REGULAR EXPRESSION TABLES" or "TCP-BASED TABLES".

CASE FOLDING

The search string is folded to lowercase before database lookup. As of Postfix 2.3, the search string is not case folded with database types such as regexp: or pcre: whose lookup fields can match both upper and lower case.

TABLE FORMAT

The input format for the **postmap**(1) command is as follows:

pattern action

When *pattern* matches a mail address, domain or host address, perform the corresponding *action*.

blank lines and comments

Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

multi-line text

A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

EMAIL ADDRESS PATTERNS

With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, patterns are tried in the order as listed below:

user@domain

Matches the specified mail address.

domain.tld

Matches *domain.tld* as the domain part of an email address.

The pattern *domain.tld* also matches subdomains, but only when the string **smtpd_access_maps** is listed in the Postfix **parent_domain_matches_subdomains** configuration setting (note that this is the default for some versions of Postfix). Otherwise, specify *.domain.tld* (note the initial dot) in

order to match subdomains.

user@ Matches all mail addresses with the specified user part.

Note: lookup of the null sender address is not possible with some types of lookup table. By default, Postfix uses <> as the lookup key for such addresses. The value is specified with the **smtpd_null_access_lookup_key** parameter in the Postfix **main.cf** file.

EMAIL ADDRESS EXTENSION

When a mail address localpart contains the optional recipient delimiter (e.g., *user+foo@domain*), the lookup order becomes: *user+foo@domain*, *user@domain*, *domain*, *user+foo@*, and *user@*.

HOST NAME/ADDRESS PATTERNS

With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, the following lookup patterns are examined in the order as listed:

domain.tld

Matches *domain.tld*.

The pattern *domain.tld* also matches subdomains, but only when the string **smtpd_access_maps** is listed in the Postfix **parent_domain_matches_subdomains** configuration setting. Otherwise, specify *.domain.tld* (note the initial dot) in order to match subdomains.

net.work.addr.ess

net.work.addr

net.work

net Matches the specified IPv4 host address or subnetwork. An IPv4 host address is a sequence of four decimal octets separated by ".".

Subnetworks are matched by repeatedly truncating the last ".octet" from the remote IPv4 host address string until a match is found in the access table, or until further truncation is not possible.

NOTE 1: The access map lookup key must be in canonical form: do not specify unnecessary null characters, and do not enclose network address information with "[]" characters.

NOTE 2: use the **cidr** lookup table type to specify network/netmask patterns. See **cidr_table**(5) for details.

net:work:addr:ess

net:work:addr

net:work

net Matches the specified IPv6 host address or subnetwork. An IPv6 host address is a sequence of three to eight hexadecimal octet pairs separated by ":".

Subnetworks are matched by repeatedly truncating the last ":octetpair" from the remote IPv6 host address string until a match is found in the access table, or until further truncation is not possible.

NOTE 1: the truncation and comparison are done with the string representation of the IPv6 host address. Thus, not all the ":" subnetworks will be tried.

NOTE 2: The access map lookup key must be in canonical form: do not specify unnecessary null characters, and do not enclose network address information with "[]" characters.

NOTE 3: use the **cidr** lookup table type to specify network/netmask patterns. See **cidr_table**(5) for details.

IPv6 support is available in Postfix 2.2 and later.

ACCEPT ACTIONS

OK Accept the address etc. that matches the pattern.

all-numerical

An all-numerical result is treated as OK. This format is generated by address-based relay authorization schemes such as pop-before-smtp.

REJECT ACTIONS

Postfix version 2.3 and later support enhanced status codes as defined in RFC 3463. When no code is specified at the beginning of the *text* below, Postfix inserts a default enhanced status code of "5.7.1" in the case of reject actions, and "4.7.1" in the case of defer actions. See "ENHANCED STATUS CODES" below.

4*NN text*

5*NN text*

Reject the address etc. that matches the pattern, and respond with the numerical three-digit code and text. **4***NN* means "try again later", while **5***NN* means "do not try again".

The reply code "421" causes Postfix to disconnect immediately (Postfix version 2.3 and later).

REJECT *optional text...*

Reject the address etc. that matches the pattern. Reply with *$reject_code optional text...* when the optional text is specified, otherwise reply with a generic error response message.

DEFER_IF_REJECT *optional text...*

Defer the request if some later restriction would result in a REJECT action. Reply with "**450 4.7.1** *optional text...* when the optional text is specified, otherwise reply with a generic error response message.

This feature is available in Postfix 2.1 and later.

DEFER_IF_PERMIT *optional text...*

Defer the request if some later restriction would result in a an explicit or implicit PERMIT action. Reply with "**450 4.7.1** *optional text...* when the optional text is specified, otherwise reply with a generic error response message.

This feature is available in Postfix 2.1 and later.

OTHER ACTIONS

restriction...

Apply the named UCE restriction(s) (**permit**, **reject**, **reject_unauth_destination**, and so on).

DISCARD *optional text...*

Claim successful delivery and silently discard the message. Log the optional text if specified, otherwise log a generic message.

Note: this action currently affects all recipients of the message. To discard only one recipient without discarding the entire message, use the transport(5) table to direct mail to the discard(8) service.

This feature is available in Postfix 2.0 and later.

DUNNO

Pretend that the lookup key was not found. This prevents Postfix from trying substrings of the lookup key (such as a subdomain name, or a network address subnetwork).

This feature is available in Postfix 2.0 and later.

FILTER *transport:destination*
After the message is queued, send the entire message through the specified external content filter. The *transport:destination* syntax is described in the **transport**(5) manual page. More information about external content filters is in the Postfix FILTER_README file.

Note: this action overrides the **content_filter** setting, and currently affects all recipients of the message.

This feature is available in Postfix 2.0 and later.

HOLD *optional text...*
Place the message on the **hold** queue, where it will sit until someone either deletes it or releases it for delivery. Log the optional text if specified, otherwise log a generic message.

Mail that is placed on hold can be examined with the **postcat**(1) command, and can be destroyed or released with the **postsuper**(1) command.

Note: use "**postsuper -r**" to release mail that was kept on hold for a significant fraction of **$maximal_queue_lifetime** or **$bounce_queue_lifetime**, or longer. Use "**postsuper -H**" only for mail that will not expire within a few delivery attempts.

Note: this action currently affects all recipients of the message.

This feature is available in Postfix 2.0 and later.

PREPEND *headername: headervalue*
Prepend the specified message header to the message. When more than one PREPEND action executes, the first prepended header appears before the second etc. prepended header.

Note: this action must execute before the message content is received; it cannot execute in the context of **smtpd_end_of_data_restrictions**.

This feature is available in Postfix 2.1 and later.

REDIRECT *user@domain*
After the message is queued, send the message to the specified address instead of the intended recipient(s).

Note: this action overrides the FILTER action, and currently affects all recipients of the message.

This feature is available in Postfix 2.1 and later.

WARN *optional text...*
Log a warning with the optional text, together with client information and if available, with helo, sender, recipient and protocol information.

This feature is available in Postfix 2.1 and later.

ENHANCED STATUS CODES

Postfix version 2.3 and later support enhanced status codes as defined in RFC 3463. When an enhanced status code is specified in an access table, it is subject to modification. The following transformations are needed when the same access table is used for client, helo, sender, or recipient access restrictions; they happen regardless of whether Postfix replies to a MAIL FROM, RCPT TO or other SMTP command.

- When a sender address matches a REJECT action, the Postfix SMTP server will transform a recipient DSN status (e.g., 4.1.1-4.1.6) into the corresponding sender DSN status, and vice versa.
- When non-address information matches a REJECT action (such as the HELO command argument or the client hostname/address), the Postfix SMTP server will transform a sender or recipient DSN

status into a generic non-address DSN status (e.g., 4.0.0).

REGULAR EXPRESSION TABLES

This section describes how the table lookups change when the table is given in the form of regular expressions. For a description of regular expression lookup table syntax, see **regexp_table**(5) or **pcre_table**(5).

Each pattern is a regular expression that is applied to the entire string being looked up. Depending on the application, that string is an entire client hostname, an entire client IP address, or an entire mail address. Thus, no parent domain or parent network search is done, *user@domain* mail addresses are not broken up into their *user@* and *domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Patterns are applied in the order as specified in the table, until a pattern is found that matches the search string.

Actions are the same as with indexed file lookups, with the additional feature that parenthesized substrings from the pattern can be interpolated as **$1**, **$2** and so on.

TCP-BASED TABLES

This section describes how the table lookups change when lookups are directed to a TCP-based server. For a description of the TCP client/server lookup protocol, see **tcp_table**(5). This feature is not available up to and including Postfix version 2.4.

Each lookup operation uses the entire query string once. Depending on the application, that string is an entire client hostname, an entire client IP address, or an entire mail address. Thus, no parent domain or parent network search is done, *user@domain* mail addresses are not broken up into their *user@* and *domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Actions are the same as with indexed file lookups.

EXAMPLE

The following example uses an indexed file, so that the order of table entries does not matter. The example permits access by the client at address 1.2.3.4 but rejects all other clients in 1.2.3.0/24. Instead of **hash** lookup tables, some systems use **dbm**. Use the command "**postconf -m**" to find out what lookup tables Postfix supports on your system.

```
/etc/postfix/main.cf:
    smtpd_client_restrictions =
        check_client_access hash:/etc/postfix/access

/etc/postfix/access:
    1.2.3   REJECT
    1.2.3.4 OK
```

Execute the command "**postmap /etc/postfix/access**" after editing the file.

BUGS

The table format does not understand quoting conventions.

SEE ALSO

postmap(1), Postfix lookup table manager
smtpd(8), SMTP server
postconf(5), configuration parameters
transport(5), transport:nexthop syntax

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
SMTPD_ACCESS_README, built-in SMTP server access control
DATABASE_README, Postfix lookup table overview

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

acct — execution accounting file

SYNOPSIS

```
#include <sys/acct.h>
```

DESCRIPTION

The kernel maintains the following *acct* information structure for all processes. If a process terminates, and accounting is enabled, the kernel calls the acct(2) function call to prepare and append the record to the accounting file.

```
/*
 * Accounting structures; these use a comp_t type which is a 3 bits base 8
 * exponent, 13 bit fraction ''floating point'' number.  Units are 1/AHZ
 * seconds.
 */
typedef u_short comp_t;

struct acct {
        char    ac_comm[10];   /* name of command */
        comp_t  ac_utime;      /* user time */
        comp_t  ac_stime;      /* system time */
        comp_t  ac_etime;      /* elapsed time */
        time_t  ac_btime;      /* starting time */
        uid_t   ac_uid;        /* user id */
        gid_t   ac_gid;        /* group id */
        short   ac_mem;        /* memory usage average */
        comp_t  ac_io;         /* count of IO blocks */
        dev_t   ac_tty;        /* controlling tty */
#define AFORK   0x01           /* forked but not execed */
#define ASU     0x02           /* used super-user permissions */
#define ACOMPAT 0x04           /* used compatibility mode */
#define ACORE   0x08           /* dumped core */
#define AXSIG   0x10           /* killed by a signal */
        char    ac_flag;       /* accounting flags */
};

/*
 * 1/AHZ is the granularity of the data encoded in the comp_t fields.
 * This is not necessarily equal to hz.
 */
#define AHZ     64

#ifdef KERNEL
struct vnode   *acctp;
#endif
```

If a terminated process was created by an execve(2), the name of the executed file (at most ten characters of it) is saved in the field *ac_comm* and its status is saved by setting one of more of the following flags in *ac_flag*: AFORK, ACORE and ASIG.

The ASU and ACOMPAT flags are no longer recorded by the system, but are retained for source compatibility.

SEE ALSO

`lastcomm`(1), `acct`(2), `execve`(2), `accton`(8), `sa`(8)

HISTORY

A **`acct`** file format appeared in Version 7 AT&T UNIX.

NAME

aliases − Postfix local alias database format

SYNOPSIS

newaliases

DESCRIPTION

The **aliases**(5) table provides a system-wide mechanism to redirect mail for local recipients. The redirections are processed by the Postfix **local**(8) delivery agent.

Normally, the **aliases**(5) table is specified as a text file that serves as input to the **postalias**(1) command. The result, an indexed file in **dbm** or **db** format, is used for fast lookup by the mail system. Execute the command **newaliases** in order to rebuild the indexed file after changing the Postfix alias database.

The input and output file formats are expected to be compatible with Sendmail version 8, and are expected to be suitable for the use as NIS maps.

Users can control delivery of their own mail by setting up **.forward** files in their home directory. Lines in per-user **.forward** files have the same syntax as the right-hand side of **aliases**(5) entries.

The format of the alias database input file is as follows:

- An alias definition has the form

  name: *value1*, *value2*, ...

- Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a '#'.
- A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

The *name* is a local address (no domain part). Use double quotes when the name contains any special characters such as whitespace, '#', ':', or '@'. The *name* is folded to lowercase, in order to make database lookups case insensitive.

In addition, when an alias exists for **owner-***name*, delivery diagnostics are directed to that address, instead of to the originator of the message. This is typically used to direct delivery errors to the maintainer of a mailing list, who is in a better position to deal with mailing list delivery problems than the originator of the undelivered mail.

The *value* contains one or more of the following:

address Mail is forwarded to *address*, which is compatible with the RFC 822 standard.

/file/name
: Mail is appended to */file/name*. See **local**(8) for details of delivery to file. Delivery is not limited to regular files. For example, to dispose of unwanted mail, deflect it to **/dev/null**.

|*command*
: Mail is piped into *command*. Commands that contain special characters, such as whitespace, should be enclosed between double quotes. See **local**(8) for details of delivery to command.

  When the command fails, a limited amount of command output is mailed back to the sender. The file **/usr/include/sysexits.h** defines the expected exit status codes. For example, use **"|exit 67"** to simulate a "user unknown" error, and **"|exit 0"** to implement an expensive black hole.

:include:*/file/name*
: Mail is sent to the destinations listed in the named file. Lines in **:include:** files have the same syntax as the right-hand side of alias entries.

  A destination can be any destination that is described in this manual page. However, delivery to

"|*command*" and */file/name* is disallowed by default. To enable, edit the **allow_mail_to_commands** and **allow_mail_to_files** configuration parameters.

ADDRESS EXTENSION

When alias database search fails, and the recipient localpart contains the optional recipient delimiter (e.g., *user+foo*), the search is repeated for the unextended address (e.g., *user*).

The **propagate_unmatched_extensions** parameter controls whether an unmatched address extension (*+foo*) is propagated to the result of table lookup.

CASE FOLDING

The local(8) delivery agent always folds the search string to lowercase before database lookup.

SECURITY

The **local**(8) delivery agent disallows regular expression substitution of $1 etc. in **alias_maps**, because that would open a security hole.

The **local**(8) delivery agent will silently ignore requests to use the **proxymap**(8) server within **alias_maps**. Instead it will open the table directly. Before Postfix version 2.2, the **local**(8) delivery agent will terminate with a fatal error.

CONFIGURATION PARAMETERS

The following **main.cf** parameters are especially relevant. The text below provides only a parameter summary. See **postconf**(5) for more details including examples.

alias_database
List of alias databases that are updated by the **newaliases**(1) command.

alias_maps
List of alias databases queried by the **local**(8) delivery agent.

allow_mail_to_commands
Restrict the usage of mail delivery to external command.

allow_mail_to_files
Restrict the usage of mail delivery to external file.

expand_owner_alias
When delivering to an alias that has an **owner-** companion alias, set the envelope sender address to the right-hand side of the owner alias, instead using of the left-hand side address.

propagate_unmatched_extensions
A list of address rewriting or forwarding mechanisms that propagate an address extension from the original address to the result. Specify zero or more of **canonical**, **virtual**, **alias**, **forward**, **include**, or **generic**.

owner_request_special
Give special treatment to **owner-***listname* and *listname***-request** addresses.

recipient_delimiter
Delimiter that separates recipients from address extensions.

Available in Postfix version 2.3 and later:

frozen_delivered_to
Update the local(8) delivery agent's Delivered-To: address (see prepend_delivered_header) only once, at the start of a delivery; do not update the Delivered-To: address while expanding aliases or .forward files.

STANDARDS

RFC 822 (ARPA Internet Text Messages)

SEE ALSO

local(8), local delivery agent
newaliases(1), create/update alias database
postalias(1), create/update alias database
postconf(5), configuration parameters

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

altq.conf — ALTQ configuration file

DESCRIPTION

The **altq.conf** file contains a number of lines specifying the behavior of queueing disciplines. Comments start with a # and extend to the end of the line.

The altqd(8) program reads /etc/altq.conf at startup and sets up queueing disciplines. BLUE, CBQ (Class-Based Queueing), FIFOQ (First-In First-Out Queue), HFSC (Hierarchical Fair Service Curve), PRIQ (Priority Queueing), RED (Random Early Detection), RIO (RED with IN/OUT), WFQ (Weighted Fair Queueing), JoBS (Joint Buffer Management and Scheduling) and CDNR (Diffserv Traffic Conditioner) can be configured in this file.

Interface Commands

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*] [*sched_type*] [*discipline-specific-options*]

The **interface** command specifies a network interface to be under control of ALTQ. One interface specification is provided for each network interface under control of ALTQ. A system configured as a router may have multiple interface specifications.

if_name
: specifies the name of a network interface (e.g., fxp0).

bandwidth
: specifies the interface bandwidth in bits per second. This is the maximum rate that the queueing discipline will allow on this interface.

tbrsize
: specifies the bucket size of a token bucket regulator in bytes. When **tbrsize** is omitted, the system automatically sets the bucket size using heuristics. The token rate is set to the interface bandwidth specified by the **interface** command.

sched_type
: Type of a queueing discipline. It must be either **blue**, **cbq**, **fifoq**, **hfsc**, **jobs**, **priq**, **red**, **rio**, or **wfq**. If the interface has only traffic conditioners and no queueing discipline, *sched_type* can be omitted.

Class Command

class *sched_type* *if_name* *class_name* *parent_name* [**red|rio**] [**ecn**] [**cleardscp**] [*discipline-specific-options*]

The **class** command specifies a packet scheduling class for CBQ, HFSC, JoBS or PRIQ. A class specifier must be provided for each packet scheduling class.

sched_type
: Type of queueing discipline. Must correspond to the discipline name in interface specification.

if_name
: Interface name. Must correspond to name in interface specification.

class_name
: Arbitrary name for this class. Must be unique for this interface.

parent_name
The name of the parent class for this class (for CBQ or HFSC). Parent class must have been previously defined. PRIQ and JoBS do not have class hierarchy and parent_name must be NULL for PRIQ and JoBS classes.

red Use RED (Random Early Detection) on this class queue. RED drops packets with the probability proportional to the average queue length.

rio Use RIO (RED with In/Out bit) on this class queue. RIO runs triple RED algorithms at the same time.

ecn Use RED/ECN (Explicit Congestion Notification) on this class queue (experimental implementation). ECN implies RED.

cleardscp
Clear diffserv codepoint in the IP header.

Filter Commands

filter *if_name class_name* [**name** *fltr_name*] [**ruleno** *num*] *filter_values*

The **filter** command specifies a filter to classify packets into a scheduling class. A filter specifier determines any statically-defined packet classification rules.

if_name Name of a network interface (e.g., fxp0).

class_name
Name of a class or a conditioner to which matching packets are directed.

name Add an arbitrary name to the filter for a future reference.

ruleno Specifies explicit order of filter matching. Filter matching is performed from a filter with a larger ruleno. Default is 0.

filter_value should be in the following format:

filter_values: *dst_addr* [**netmask** *mask*] *dport* *src_addr* [**netmask** *mask*] *sport* *proto* [**tos** *value* [**tosmask** *value*]] [**gpi** *value*]

Here *dst_addr* and *src_addr* are dotted-decimal addresses of the destination and the source respectively. An address may be followed by **netmask** keyword. *dport* and *sport* are port numbers of the destination and the source respectively. *proto* is a protocol number defined for IP packets (e.g. 6 for TCP). **tos** keyword can be used to specify the type of service field value. **gpi** keyword can be used to specify the Security Parameter Index value for IPsec.

When filter value 0 is used, it is taken as a wildcard.

filter6 *if_name class_name* [**name** *fltr_name*] [**ruleno** *num*] *filter6_values*

The **filter6** command is for IPv6. *filter6_value* should be in the following format:

filter6_values:
dst_addr[/prefix_len] *dport* *src_addr*[/prefix_len] *sport* *proto* [**flowlabel** *value*] [**tclass** *value* [**tclassmask** *value*]] [**gpi** *value*]

Here *dst_addr* and *src_addr* are IPv6 addresses of the destination and the source respectively. An address may be followed by an optional address prefix length. *dport* and *sport* are port numbers of the destination and the source respectively. *proto* is a protocol number defined for IPv6 packets (e.g. 6 for TCP). **flowlabel** keyword can be used to specify the flowlabel field value. **tclass** keyword can be used to specify the traffic class field value. **gpi** keyword can be used to specify the Security Parameter Index value for IPsec.

When filter value 0 is used, it is taken as a wildcard.

CBQ Commands

CBQ (Class Based Queueing) achieves both partitioning and sharing of link bandwidth by hierarchically structured classes. Each class has its own queue and is assigned its share of bandwidth. A child class can borrow bandwidth from its parent class as long as excess bandwidth is available.

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*] [*sched_type*] [**efficient**]

if_name
: specifies the name of a network interface (e.g., fxp0).

bandwidth
: specifies the interface bandwidth in bits per second.

tbrsize
: specifies the bucket size of a token bucket regulator in bytes.

sched_type
: must be either **cbq**, **cbq-wrr** (weighted-round robin) or **cbq-prr** (packet-by-packet round robin). **cbq** is equivalent to **cbq-wrr**.

efficient
: Enables CBQ's link efficiency mode. This means that the scheduler will send a packet from the first overlimit class it encounters of all classes of the link-sharing structure when all classes are overlimit. This will also cause the scheduler to use greater than its assigned bandwidth, if the link is capable of more than the assigned bandwidth. By default, this mode is turned off. By adding the keyword **efficient** to the interface specification line, enables this mode.

class *sched_type if_name class_name parent_name* [**admission cntlload|none**] [**priority** *pri*] [**pbandwidth** *percent*] [**exactbandwidth** *bps*] [**borrow**] [**default**] [**control**] [**maxburst** *count*] [**minburst** *count*] [**maxdelay** *msec*] [**packetsize** *bytes*] [**maxpacketsize** *bytes*] [**red|rio**] [**ecn**] [**flowvalve**] [**cleardscp**]

The **class** command specifies a CBQ class. The classes are organized as a hierarchy, and every class, except for the root class, has a parent.

sched_type
: must be **cbq** for a CBQ class.

if_name
: Interface name. Must correspond to name in interface specification.

class_name
: Arbitrary name for this class. Must be unique within the class hierarchy for this interface. The name **ctl_class** is a reserved class name.

parent_name
: The name of the parent class for this class or NULL if this is the root class. Parent class must have been previously defined.

admission
: The type of admission control and QoS type. **cntlload** is controlled load service for RSVP, otherwise, it should be **none**. The default is **none**.

priority
: High numbers are higher priority. Max value is 7 and Min value is 0. Default is 1.

pbandwidth
The percentage of the interface bandwidth allocated to this class. Generally should add up to 100 percent at each level of the class hierarchy, although other amounts can be specified for purposes of experimentation.

exactbandwidth
Specify the bandwidth in bits-per-second instead of **pbandwidth**. Note that the bandwidth allocation of CBQ is not so precise but this is just a way to pass a parameter to CBQ; the user is supposed to know the detailed internals of CBQ. **pbandwidth** is a preferred way to specify the bandwidth of a class.

borrow The class can borrow bandwidth from its parent class when this class is overlimit. If this keyword is not present, then no borrowing is done, and the packet is delayed or dropped when the class is overlimit.

default
Specify the default class. When this keyword is present, all packets that do not match some classification criteria are assigned to this class. Must be exactly one class on each interface defined as the default class.

control
Specify the control class. When this keyword is present, the predefined control class packets (RSVP, IGMP, and ICMP) are assigned to this class. Note that when the control class is not specified by the time the default class is created, one is automatically created with default parameters. Thus, if the control class is specified, it must be listed before the default class. Must be exactly one class on each interface defined as the control class.

maxburst
The maximum burst of back-to-back packets allowed in this class. Default is 16 but the default value is automatically reduced to 4 when the class bandwidth is small (about less than 1Mbps).

minburst
The minimum burst is used to obtain the steady state burst size. It's the parameter to help compute offtime for the class. Offtime is the amount of time a class is to wait between packets. Default is 2.

maxdelay
The maxdelay is specified in milliseconds and used to obtain the max queue size of the class. If not specified, the default max queue size (30 packets) is used.

packetsize
The average packet size in bytes to be used in CBQ over-/under-limit computations. Default value is MTU of the interface.

maxpacketsize
The maximum packet size in bytes for the class. Default value is MTU of the interface.

red enables RED on this class queue.

rio enables RIO on this class queue.

ecn enables RED/ECN on this class queue.

flowvalve
enables RED/flow-valve (a.k.a. red-penalty-box) on this class queue.

cleardscp
clears diffserv codepoint in the IP header.

HFSC Commands

HFSC (Hierarchical Fair Service Curve) supports both link-sharing and guaranteed real-time services. H-FSC employs a service curve based QoS model, and its unique feature is an ability to decouple delay and bandwidth allocation. HFSC has 2 independent scheduling mechanisms. Real-time scheduling is used to guarantee the delay and the bandwidth allocation at the same time. Hierarchical link-sharing is used to distribute the excess bandwidth. When dequeueing a packet, HFSC always tries real-time scheduling first. If no packet is eligible for real-time scheduling, link-sharing scheduling is performed. HFSC does not use class hierarchy for real-time scheduling. Additionally, an upper-limit service curve can be specified for link-sharing to set the upper limit allowed for the class.

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*] [*sched_type*]

if_name
specifies the name of a network interface (e.g., fxp0).

bandwidth
specifies the interface bandwidth in bits per second.

tbrsize
specifies the bucket size of a token bucket regulator in bytes.

sched_type
must be **hfsc** for HFSC.

class *sched_type if_name class_name parent_name* [**admission cntlload|none**] [[**sc** *m1 d m2*]] [[**rt** *m1 d m2*]] [[**ls** *m1 d m2*]] [[**ul** *m1 d m2*]] [**pshare** *percent*] [**grate** *bps*] [**bandwidth** *bps*] [**ulimit** *bps*] [**default**] [**qlimit** *count*] [**red|rio**] [**ecn**] [**cleardscp**]

The **class** command specifies a HFSC class. The classes are organized as a hierarchy, and every class, except for the root class, has a parent.

Each HFSC class has 2 service curves, the real-time service curve and the link-sharing service curve. Service curves are specified by [*type m1 d m2*]. *type* should be either **sc**, **rt**, **ls**, or **ul**. **sc** (service curve) is used to set the same values to both real-time and link-sharing service curves. **rt** (real-time) is used to specify the real-time service curve. **ls** (link-sharing) is used to specify the link-sharing service curve. **ul** (upper-limit) is used to specify the upper-limit service curve for link-sharing. *m1* is the slope of the first segment specified in bits-per-second. *d* is the x-projection of the intersection point of the 2 segments specified in milliseconds. *m2* is the slope of the second segment specified in bits-per-second.

sched_type
must be **hfsc** for a HFSC class.

if_name
Interface name. Must correspond to name in interface specification.

class_name
Arbitrary name for this class. Must be unique within the class hierarchy for this interface. The name **root** is a reserved class name for the root class. The root class for the interface is automatically created by the **interface** command.

parent_name
The name of the parent class for this class. Keyword **root** is used when the parent is the root class. Parent class must have been previously defined.

admission
The type of admission control and QoS type. **cntlload** is controlled load service for RSVP, otherwise, it should be **none**. The default is **none**.

pshare Percent of the link share. This specifies a linear link-sharing service curve as a fraction of the link bandwidth. It is a short hand of `[ls 0 0 (link-bandwidth * percent / 100)]`.

grate Guaranteed rate. This specifies a linear real-time service curve. It is a short hand of `[rt 0 0 bps]`.

bandwidth
This is a short hand of `[sc 0 0 bps]`.

ulimit Upper limit rate. This specifies a upper-limit service curve. It is a short hand of `[ul 0 0 bps]`.

default
Specify the default class. When this keyword is present, all packets that do not match some classification criteria are assigned to this class. Must be exactly one class on each interface defined as the default class.

qlimit The maximum queue size in number of packets. Default value is 50.

red enables RED on this class queue.

rio enables RIO on this class queue.

ecn enables RED/ECN on this class queue.

cleardscp
clears diffserv codepoint in the IP header.

PRIQ Commands

PRIQ (Priority Queueing) implements a simple priority-based queueing. A higher priority class is always served first. Up to 16 priorities can be used with PRIQ.

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*] [*sched_type*]

if_name
specifies the name of a network interface (e.g., fxp0).

bandwidth
specifies the interface bandwidth in bits per second.

tbrsize
specifies the bucket size of a token bucket regulator in bytes.

sched_type
must be **priq** for PRIQ.

class *sched_type if_name class_name parent_name* [**priority** *pri*] [**default**] [**qlimit** *count*] [**red|rio**] [**ecn**] [**cleardscp**]

sched_type
must be **priq** for a PRIQ class.

if_name
Interface name. Must correspond to name in interface specification.

class_name
Arbitrary name for this class. Must be unique for this interface.

parent_name
Parent class must be `NULL` for PRIQ.

priority
High numbers are higher priority. Max value is 15 and Min value is 0. Default is 0. A higher priority class is always served first in PRIQ. Priority must be unique for the interface.

default
Specify the default class. When this keyword is present, all packets that do not match some classification criteria are assigned to this class. Must be exactly one class on each interface defined as the default class.

qlimit The maximum queue size in number of packets. Default value is 50.

red enables RED on this class queue.

rio enables RIO on this class queue.

ecn enables RED/ECN on this class queue.

cleardscp
clears diffserv codepoint in the IP header.

WFQ Commands
WFQ (Weighted Fair Queueing) implements a weighted-round robin scheduler for a set of queue. A weight can be assigned to each queue to give a different proportion of the link capacity. A hash function is used to map a flow to one of a set of queues, and thus, it is possible for two different flows to be mapped into the same queue.

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*] [*sched_type*] [**nqueues** *count*][**qsize** *bytes*][**hash** *policy*]

if_name
specifies the name of a network interface (e.g., fxp0).

bandwidth
specifies the interface bandwidth in bits per second.

tbrsize
specifies the bucket size of a token bucket regulator in bytes.

sched_type
must be **wfq** for WFQ.

nqueues
The number of queues in WFQ. Default value is 256.

qsize The size of each queue in number of bytes. Default value is 64K bytes.

hash Type of hash policy to select a queue. **dstaddr** specifies a hashing policy by IP destination address. **full** specifies a hashing policy by IP addresses and ports. **srcport** specifies a hashing policy by IP source port number. Default is **dstaddr**

FIFOQ Commands
FIFOQ (First-In First-Out Queueing) is a simple tail-drop FIFO queue. FIFOQ is the simplest possible implementation of a queueing discipline in ALTQ, and can be used to compare with other queueing disciplines. FIFOQ can be also used as a template for those who want to write their own queueing disciplines.

interface *if_name* [**bandwidth** *bps*][**tbrsize** *bytes*][*sched_type*][**qlimit** *count*]

if_name
specifies the name of a network interface (e.g., fxp0).

bandwidth
specifies the interface bandwidth in bits per second.

tbrsize
specifies the bucket size of a token bucket regulator in bytes.

sched_type
must be **fifoq** for FIFOQ.

qlimit The maximum queue size in number of packets. Default value is 50.

RED Commands
RED (Random Early Detection) is an implicit congestion notification mechanism that exercises packet dropping or packet marking stochastically according to the average queue length. RED can be viewed as a buffer management mechanism and can be integrated into other packet scheduling schemes.

red *min_th* *max_th* *inv_pmax*

The **red** command sets the default RED parameters. *min_th* and *max_th* are the minimum and the maximum threshold values. *inv_pmax* is the inverse (reciprocal) of the maximum drop probability. For example, 10 means the maximum drop probability of 1/10.

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*] [*sched_type*] [**qlimit** *count*] [**packetsize** *bytes*] [**weight** *n*] [**thmin** *n*] [**thmax** *n*] [**invpmax** *n*] [**ecn**] [**flowvalve**]

if_name
specifies the name of a network interface (e.g., fxp0).

bandwidth
specifies the interface bandwidth in bits per second.

tbrsize
specifies the bucket size of a token bucket regulator in bytes.

sched_type
must be **red** for RED.

qlimit The maximum queue size in number of packets. Default value is 60.

packetsize
The average packet size in number of bytes. This parameter is used to calibrate the idle period. Default value is 1000.

weight The inverse of the weight of EWMA (exponentially weighted moving average).

thmin The minimum threshold.

thmax The maximum threshold.

invpmax
The inverse of the maximum drop probability.

ecn enables ECN.

flowvalve
enables flowvalve.

RIO Commands

ALTQ/RIO has 3 drop precedence levels defined for the Assured Forwarding of DiffServ (RFC2597). Since adaptive flows are likely to stay under the medium drop precedence level under congestion, the medium drop precedence would protect adaptive flows from unadaptive flows.

The original RIO has 2 sets of RED parameters; one for in-profile packets and the other for out-of-profile packets. At the ingress of the network, profile meters tag packets as IN or OUT based on contracted profiles for customers. Inside the network, IN packets receive preferential treatment by the RIO dropper. It is possible to provision the network not to drop IN packets at all by providing enough capacity for the total volume of IN packets. Thus, RIO can be used to provide a service that statistically assures capacity allocated for users. This mechanism can be extended to support an arbitrary number of drop precedence levels. ALTQ supports 3 drop precedence levels.

rio *low_min_th low_max_th low_inv_pmax medium_min_th medium_max_th medium_inv_pmax high_min_th high_max_th high_inv_pmax*

The **rio** command sets the default RIO parameters. The parameters are RED parameters for 3 (low, medium, high) drop precedence.

interface *if_name* [**bandwidth** *bps*][**tbrsize** *bytes*][*sched_type*][**qlimit** *count*] [**packetsize** *bytes*][**weight** *n*][**lo_thmin** *n*][**lo_thmax** *n*][**lo_invpmax** *n*] [**med_thmin** *n*] [**med_thmax** *n*] [**med_invpmax** *n*] [**hi_thmin** *n*] [**hi_thmax** *n*][**hi_invpmax** *n*][**ecn**]

if_name
: specifies the name of a network interface (e.g., fxp0).

bandwidth
: specifies the interface bandwidth in bits per second.

tbrsize
: specifies the bucket size of a token bucket regulator in bytes.

sched_type
: must be **rio** for RIO.

qlimit The maximum queue size in number of packets. Default value is 60.

packetsize
: The average packet size in number of bytes. This parameter is used to calibrate the idle period. Default value is 1000.

weight The inverse of the weight of EWMA (exponentially weighted moving average).

lo_thmin
: The minimum threshold for low drop precedence.

lo_thmax
: The maximum threshold for low drop precedence.

lo_invpmax
: The inverse of the maximum drop probability for low drop precedence.

med_thmin
: The minimum threshold for medium drop precedence.

med_thmax
: The maximum threshold for medium drop precedence.

med_invpmax
The inverse of the maximum drop probability for medium drop precedence.

hi_thmin
The minimum threshold for high drop precedence.

hi_thmax
The maximum threshold for high drop precedence.

hi_invpmax
The inverse of the maximum drop probability for high drop precedence.

ecn enables ECN.

BLUE Commands

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*] [*sched_type*] [**qlimit** *count*] [**packetsize** *bytes*] [**maxpmark** *n*] [**holdtime** *usec*] [**ecn**]

if_name
specifies the name of a network interface (e.g., fxp0).

bandwidth
specifies the interface bandwidth in bits per second.

tbrsize
specifies the bucket size of a token bucket regulator in bytes.

sched_type
must be **blue** for BLUE.

qlimit The maximum queue size in number of packets. Default value is 60.

packetsize
The average packet size in number of bytes. Default value is 1000.

maxpmark
specifies the precision of marking probability.

holdtime
specifies the hold time in usec.

ecn enables ECN.

CDNR Commands

The **conditioner** command specifies a diffserv traffic conditioner. A traffic conditioner is not a queueing discipline but a component to meter, mark or drop incoming packets according to some rules.

As opposed to a queueing discipline, a traffic conditioner handles incoming packets at an input interface. If no queueing discipline (e.g., CBQ) is used for the interface, a null interface command should be used to specify an input network interface.

interface *if_name* [**bandwidth** *bps*] [**tbrsize** *bytes*]

The **conditioner** command has the following syntax.

conditioner *if_name cdnr_name* ⟨action⟩

if_name Interface name. Must correspond to name in interface specification.

cdnr_name
Arbitrary name for this conditioner. Must be unique for this interface.

action Action of the conditioner.

An action can be a recursively defined action. The following actions are defined.

pass

pass allows the packet to go through without any modification to the packet.

drop

drop rejects the packet. The packet is immediately discarded.

mark *value*

mark sets the specified value to the ds field in the IP header. Then, the packet is allowed to go through.

tbmeter *rate depth* ⟨in_action⟩ ⟨out_action⟩

tbmeter is a token bucket meter configured with rate and depth parameters. Rate is token rate in bits-per-second. Depth is bucket depth in KB. When an incoming packet is in profile (available token is more than the packet size), tbmeter takes in_action. Otherwise, tbmeter takes out_action.

trtcm *cmtd_rate cmtd_depth peak_rate peak_depth* ⟨green_action⟩ ⟨yellow_action⟩ ⟨red_action⟩ [**coloraware|colorblind**]

trtcm is a 2-rate 3 color marker for Assured Forwarding. A trtcm consists of 2 token buckets, one for a committed rate and the other for a peak rate. When an incoming packet is in the committed profile, trtcm takes green_action. When the packet is out of the committed profile but in the peak profile, trtcm takes yellow_action. Otherwise, tbtcm takes red_action. A trtcm is either color-aware or color-blind. A color-aware trtcm do not raise the color (ds field value), that is, a yellow packet can be yellow or red but can not be blue. Default is color-blind.

tswtcm *cmtd_rate peak_rate avg_interval* ⟨green_action⟩ ⟨yellow_action⟩ ⟨red_action⟩

tswtcm is a time sliding window 3 color marker for Assured Forwarding. A tswtcm differs from trtcm in that a tswtcm probabilistically marks packets. A tswtcm consists of 2 rates, one for a committed rate and the other for a peak rate. When an incoming packet is in the committed profile, tswtcm takes green_action. When the packet is out of the committed profile but in the peak profile, tswtcm takes yellow_action. Otherwise, tswtcm takes red_action. cmtd_rate and peak_rate are specified in bits per second. avg_interval provides the size of time window for averaging incoming rate, and is specified in milliseconds. 500 msec is ok for normal settings.

JoBS Commands

JoBS (Joint Buffer Management and Scheduling) is a queuing discipline that can enforce any feasible mix of absolute and proportional guarantees on packet losses, packet delays, and throughput, for classes of traffic, on a per-hop basis. No admission control is performed, thus if the set of service guarantees becomes infeasible, some service guarantees may be relaxed.

interface *if_name* [**bandwidth** *bps*] [**qlimit** *count*] [**separate**] [**tbrsize** *bytes*] [*sched_type*]

if_name
specifies the name of a network interface (e.g., fxp0).

bandwidth
specifies the interface bandwidth in bits per second.

qlimit specifies the maximum queue size in number of packets.

separate
specifies that classes have independent buffers. The default is to have a shared buffer for all classes. If this option is specified, qlimit applies to each independent buffer.

tbrsize
specifies the bucket size of a token bucket regulator in bytes.

sched_type
must be **jobs** for JoBS.

class *sched_type if_name class_name parent_name* [**priority** *pri*][**default**][**adc** *microsecs*][**alc** *fraction*][**arc** *bps*][**rdc** *prop*][**rlc** *prop*]

sched_type
must be **jobs** for a JoBS class.

if_name
Interface name. Must correspond to name in interface specification.

class_name
Arbitrary name for this class. Must be unique for this interface.

parent_name
Parent class must be NULL for JoBS.

priority
Priority index used for proportional differentiation. Max value is 15 and Min value is 0. Default is 0. Priority must be unique for the interface.

default
Specify the default class. When this keyword is present, all packets that do not match some classification criteria are assigned to this class. Must be exactly one class on each interface defined as the default class.

adc Specifies an upper bound on delays for that class (in microseconds). A value of −1 will indicate the absence of delay bound. By default, no delay bound is offered.

alc Specifies a upper bound on loss rate for that class (in fraction of 1, for instance a 1% loss rate bound will be expressed as 0.01). A value of −1 will indicate the absence of loss rate bound. By default, no loss rate bound is offered.

arc Specifies a lower bound on the throughput received by that class (in bits per second). A value of −1 will indicate the absence of throughput bound. By default, no throughput bound is offered.

rdc Specifies a proportional delay differentiation factor between that class and the class with the successive priority index. For instance, for priority 1, an rdc of 2 specifies that the delays of packets marked as class 2 will roughly be twice the delays of packets marked as class 1. A value of −1 indicates the absence of proportional differentiation on that class. Note that class N if N is the maximum priority should have a dummy coefficient different from −1 if proportional delay differentiation is desired on Class N. By default, no proportional delay differentiation is offered.

rlc Specifies a proportional loss differentiation factor between that class and the class with the successive priority index. For instance, for priority 1, an rlc of 2 specifies that the loss rate of packets marked as class 2 will roughly be twice the loss rate of packets marked as class 1. A value of −1 indicates the absence of proportional differentiation on that class. Note that class N if N is the maximum priority should have a dummy coefficient different from −1 if proportional loss differentiation is desired on Class N. By default, no proportional loss differentiation is offered.

EXAMPLES

CBQ Example

```
#
# cbq configuration for vx0 (10Mbps ether)
# give at least 40% to TCP
# limit HTTP from network 133.138.1.0 up to 10%, use RED.
# other traffic goes into default class
#
interface vx0 bandwidth 10M cbq
#
class cbq vx0 root_class NULL priority 0 pbandwidth 100
class cbq vx0 def_class root_class borrow pbandwidth 95 default
class cbq vx0 tcp_class def_class borrow pbandwidth  40
        filter vx0 tcp_class 0 0 0 0 6
class cbq vx0 csl_class tcp_class pbandwidth  10   red
        filter vx0 csl_class 0 0 133.138.1.0 netmask 0xffffff00 80 6
        filter vx0 csl_class 133.138.1.0 netmask 0xffffff00 0 0 80 6
#
# sample filter6 command
#
                filter6 vx0 csl_class ::0 0 d000:a:0:123::/64 80 6
```

HFSC Example

```
#
# hfsc configuration for hierarchical sharing
#
interface pvc0 bandwidth 45M hfsc
#
# (10% of the bandwidth share goes to the default class)
class hfsc pvc0 def_class root pshare 10 default
#
#          bandwidth share    guaranteed rate
#    CMU:       45%             15Mbps
#    PITT:      45%             15Mbps
#
class hfsc pvc0 cmu  root pshare 45 grate 15M
class hfsc pvc0 pitt root pshare 45 grate 15M
#
# CMU      bandwidth share    guaranteed rate
#    CS:        20%             10Mbps
#    other:     20%              5Mbps
#
class hfsc pvc0 cmu_other cmu  pshare 20 grate 10M
        filter pvc0 cmu_other   0 0 128.2.0.0   netmask 0xffff0000 0 0
class hfsc pvc0 cmu_cs    cmu  pshare 20 grate  5M
```

```
        filter pvc0 cmu_cs      0 0 128.2.242.0 netmask 0xffffff00 0 0
#
# PITT     bandwidth share    guaranteed rate
#    CS:        20%             10Mbps
#    other:     20%              5Mbps
#
class hfsc pvc0 pitt_other pitt  pshare 20 grate 10M
        filter pvc0 pitt_other  0 0 136.142.0.0  netmask 0xffff0000 0 0
class hfsc pvc0 pitt_cs    pitt  pshare 20 grate  5M
        filter pvc0 pitt_cs     0 0 136.142.79.0 netmask 0xffffff00 0 0
```

HFSC Example (simpler one with ulimit)

```
#
interface fxp0 bandwidth 90M hfsc
# reserve 20% for default class
class hfsc fxp0 def_class root pshare 20 default
# shared class for TCP and UDP
class hfsc fxp0 shared_class root bandwidth 72M
# shared class for all TCP
class hfsc fxp0 tcp_shared shared_class bandwidth 40M ulimit 60M
# generic tcp
class hfsc fxp0 tcp_class tcp_shared bandwidth 15M ulimit 50M
       filter fxp0 tcp_class 0 0 0 0 6
# http
class hfsc fxp0 http_class tcp_shared bandwidth 25M ulimit 40M
       filter fxp0 http_class 0 80 0 0 6
       filter fxp0 http_class 0 0 0 80 6
# udp
class hfsc fxp0 udp_class shared_class bandwidth 15M ulimit 20M
       filter fxp0 udp_class 0 0 0 0 17
```

PRIQ Example

```
#
# priq configuration for fxp0 (100Mbps ether)
#     icmp: high priority
#     tcp:  medium priority
#     others: low priority
#
interface fxp0 bandwidth 100M priq
#
class priq fxp0 high_class NULL priority 2
       filter fxp0 high_class 0 0 0 0 1
class priq fxp0 med_class NULL priority 1
       filter fxp0 med_class 0 0 0 0 6
class priq fxp0 low_class NULL priority 0 default
```

WFQ Example

```
interface pvc0 bandwidth 134000000 wfq
```

FIFOQ Example

```
interface rl0 bandwidth 10M fifoq
```

Conditioner Example

```
#
interface fxp0
#
# a simple dropper
# discard all packets from 192.168.0.83
#
conditioner fxp0 dropper <drop>
        filter fxp0 dropper 0 0 192.168.0.83 0 0

#
# EF conditioner
# mark EF to all packets from 192.168.0.117
#
conditioner pvc1 ef_cdnr <tbmeter 6M 64K <mark 0xb8><drop>>
        filter fxp0 ef_cdnr 0 0 192.168.0.117 0 0

#
# AF1x conditioner
# mark AF1x to packets from 192.168.0.178
#       AF11 (low drop precedence):    less than 3Mbps
#       AF12 (medium drop precedence): more than 3Mbps and less than 10Mbps
#       AF13 (high drop precedence):   more than 10Mbps
#
conditioner fxp0 af1x_cdnr <trtcm 3M 32K 10M 64K <mark 0x28><mark 0x30><mark 0x38>>
        filter fxp0 af1x_cdnr 0 0 192.168.0.178 0 0
```

SEE ALSO

`altqd`(8)

BUGS

This man page is incomplete. For more information read the source.

NAME

amd.conf − Amd configuration file

SYNOPSIS

amd.conf

DESCRIPTION

The **amd.conf** file is the configuration file for Amd, as part of the am-utils suite.

amd.conf contains runtime configuration information for the **Amd** automounter program.

FILE FORMAT

The file consists of sections and parameters. A section begins with the name of the section in square brackets and continues until the next section begins or the end the file is reached. Sections contain parameters of the form 'name = value'.

The file is line-based - that is, each newline-terminated line represents either a comment, a section name or a parameter. No line-continuation syntax is available.

Section, parameter names and their values are case sensitive.

Only the first equals sign in a parameter is significant. Whitespace before or after the first equals sign is discarded. Leading, trailing and internal whitespace in section and parameter names is irrelevant. Leading and trailing whitespace in a parameter value is discarded. Internal whitespace within a parameter value is not allowed, unless the whole parameter value is quoted with double quotes as in 'name = "some value"'.

Any line beginning with a pound sign (#) is ignored, as are lines containing only whitespace.

The values following the equals sign in parameters are all either a string (no quotes needed if string does not include spaces) or a boolean, which may be given as yes/no. Case is significant in all values. Some items such as cache timeouts are numeric.

SECTIONS

The [global] section

Parameters in this section either apply to Amd as a whole, or to all other regular map sections which follow. There should be only one global section defined in one configuration file.

It is highly recommended that this section be specified first in the configuration file. If it is not, then regular map sections which precede it will not use global values defined later.

Regular [/map] sections

Parameters in regular (non-global) sections apply to a single map entry. For example, if the map section **[/homes]** is defined, then all parameters following it will be applied to the */homes* Amd-managed mount point.

PARAMETERS

Parameters common to all sections

These parameters can be specified either in the global or a map specific section. Entries specified in a map-specific section override the default value or one defined in the global section. If such a common parameter is specified only in the global section, it is applicable to all regular map sections that follow.

browsable_dirs (string, default=no)

If "yes," then Amd's top-level mount points will be browsable to **readdir**(3) calls. This means you could run for example **ls**(1) and see what keys are available to mount in that directory. Not all entries are made visible to readdir(3): the "/default" entry, wildcard entries, and those with a "/" in them are not included. If you specify "full" to this option, all but "/default" will be visible. Note that if you run a command which will attempt to **stat**(2) the entries, such as often done by "ls -l" or "ls -F," Amd will attempt to mount *every* entry in that map. This is often called a ''mount storm.''

map_defaults (string, default to empty)
This option sets a string to be used as the map's /defaults entry, overriding any /defaults specified in the map. This allows local users to override map defaults without modifying maps globally.

map_options (string, default no options)
This option is the same as specifying map options on the command line to Amd, such as "cache:=all".

map_type (string, default search all map types)
If specified, Amd will initialize the map only for the type given. This is useful to avoid the default map search type used by Amd which takes longer and can have undesired side-effects such as initializing NIS even if not used. Possible values are

exec executable maps
file plain files
hesiod Hesiod name service from MIT
ldap Lightweight Directory Access Protocol
ndbm (New) dbm style hash files
nis Network Information Services (version 2)
nisplus Network Information Services Plus (version 3)
passwd local password files
union union maps

mount_type (string, default=nfs)
All Amd mount types default to NFS. That is, Amd is an NFS server on the map mount points, for the local host it is running on. If "autofs" is specified, Amd will be an autofs server for those mount points.

autofs_use_lofs (string, default=yes)
When set to "yes" and using Autofs, Amd will use lofs-type (loopback) mounts for type:=link mounts. This has the advantage of mounting in place, and users get to the see the same pathname that they chdir'ed into. If this option is set to "no," then Amd will use symlinks instead: that code is more tested, but negates autofs's big advantage of in-place mounts.

search_path (string, default no search path)
This provides a (colon-delimited) search path for file maps. Using a search path, sites can allow for local map customizations and overrides, and can distributed maps in several locations as needed.

selectors_in_defaults (boolean, default=no)
If "yes," then the /defaults entry of maps will search for and process any selectors before setting defaults for all other keys in that map. Useful when you want to set different options for a complete map based on some parameters. For example, you may want to better the NFS performance over slow slip-based networks as follows:

```
/defaults \
   wire==slip-net;opts:=intr,rsize=1024,wsize=1024 \
   wire!=slip-net;opts:=intr,rsize=8192,wsize=8192
```

Deprecated form: selectors_on_default

Parameters applicable to the global section only

arch (string, default to compiled in value)
Same as the **−A** option to Amd. Allows you to override the value of the *arch* Amd variable.

auto_attrcache (numeric, default=0)
Specify in seconds (or units of 0.1 seconds, depending on the OS), what is the (kernel-side) NFS attribute cache timeout for @i{Amd}'s own automount points. A value of 0 is supposed to turn off attribute caching, meaning that @i{Amd} will be consulted via a kernel-RPC each time someone stat()'s the mount point (which could be abused as a denial-of-service attack). Warning: some OSs are incapable of turning off the NFS attribute cache reliably. On such systems, Amd may not work reliably under heavy load. See the README.attrcache document in the Am-utils distribution for more details.

auto_dir (string, default=/a)
Same as the **−a** option to Amd. This sets the private directory where Amd will create sub-directories for its real mount points.

cache_duration (numeric, default=300)
Same as the **−c** option to Amd. Sets the duration in seconds that looked-up or mounted map entries remain in the cache.

cluster (string, default no cluster)
Same as the **−C** option to Amd. Specifies the alternate HP-UX cluster to use.

debug_mtab_file (string, default=/tmp/mnttab)
Path to mtab file that is used by Amd to store a list of mounted file systems during debug-mtab mode. This option only applies to systems that store mtab information on disk.

debug_options (string, default no debug options)
Same as the **−D** option to Amd. Specify any debugging options for Amd. Works only if am-utils was configured for debugging using the --enable-debug option. The "mem" option, as well as all other options, can be turned on via --enable-debug=mem. Otherwise debugging options are ignored. Options are comma delimited, and can be preceded by the string "no" to negate their meaning. You can get the list of supported debugging options by running Amd −H. Possible values are:

all all options
amq register for amq
daemon enter daemon mode
fork fork server
full program trace
hrtime print high resolution time stamps (only if syslog(3) is not used)
info info service specific debugging (hesiod, nis, etc.)
mem trace memory allocations
mtab use local "./mtab" file
readdir show browsable_dirs progress
str debug string munging
test full debug but no daemon
trace trace protocol and NFS mount arguments
xdrtrace trace XDR routines

dismount_interval (numeric, default=120)
Same as the **−w** option to Amd. Specify in seconds, the time between attempts to dismount file systems that have exceeded their cached times.

domain_strip (boolean, default=yes)
If "yes," then the domain name part referred to by ${rhost} is stripped off. This is useful to keep logs and smaller. If "no," then the domain name part is left changed. This is useful when using multiple domains with the same maps (as you may have hosts whose domain-stripped name is identical).

exec_map_timeout (numeric, default=10)
The timeout in seconds that *Amd* will wait for an executable map program before an answer is returned from that program (or script). This value should be set to as small as possible while still allowing normal replies to be returned before the timer expires, because during the time that the executable map program is queried, *Amd* is essentially waiting and is thus not responding to any other queries.

forced_unmounts (boolean, default=no)
If set to "yes," and the client OS supports forced or lazy unmounts, then *Amd* will attempt to use them if it gets any of three serious error conditions when trying to unmount an existing mount point or mount on top of one: EIO, ESTALE, or EBUSY.

This could be useful to recover from serious conditions such as hardware failure of mounted disks, or NFS servers which are down permanently, were migrated, or changed their IP address. Only "type:=toplvl" mounts hung with EBUSY are forcibly unmounted using this option, which is useful to recover from a hung *Amd*).

full_os (string, default to compiled in value)
The full name of the operating system, along with its version. Allows you to override the compiled-in full name and version of the operating system. Useful when the compiled-in name is not desired. For example, the full operating system name on linux comes up as ‘‘linux’’, but you can override it to ‘‘linux-2.2.5.’’

fully_qualified_hosts (string, default=no)
If "yes," *Amd* will perform RPC authentication using fully-qualified host names. This is necessary for some systems, and especially when performing cross-domain mounting. For this function to work, the *Amd* variable ${hostd} is used, requiring that ${domain} not be null.

hesiod_base (string, default=automount)
Specify the base name for hesiod maps.

karch (string, default to karch of the system)
Same as the **−k** option to amd. Allows you to override the kernel-architecture of your system. Useful for example on Sun (SPARC) machines, where you can build one amd binary, and run it on multiple machines, yet you want each one to get the correct *karch* variable set (for example, sun4c, sun4m, sun4u, etc.) Note that if not specified, Amd will use uname(3) to figure out the kernel architecture of the machine.

ldap_base (string, default not set)
Specify the base name for LDAP. This often includes LDAP-specific values such as country and organization.

ldap_cache_maxmem (numeric, default=131072)
Specify the maximum memory Amd should use to cache LDAP entries.

ldap_cache_seconds (numeric, default=0)
Specify the number of seconds to keep entries in the cache.

ldap_hostports (string, default not set)
Specify the LDAP host and port values.

ldap_proto_version (numeric, default=2)
Specify the version of the LDAP protocol to use.

local_domain (string, default no sub-domain)
Same as the **−d** option to Amd. Specify the local domain name. If this option is not given the domain name is determined from the hostname, by removing the first component of the fully-qualified host name.

localhost_address (string, default to localhost or 127.0.0.1)
Specify the name or IP address for Amd to use when connecting the sockets for the local NFS server and the RPC server. This defaults to 127.0.0.1 or whatever the host reports as its local address. This parameter is useful on hosts with multiple addresses where you want to force Amd to connect to a specific address.

log_file (string, default=/dev/stderr)
Same as the **−l** option to Amd. Specify a file name to log Amd events to. If the string **/dev/stderr** is specified, Amd will send its events to the standard error file descriptor. If the string **syslog** is given, Amd will record its events with the system logger **syslogd**(8). The default syslog facility used is LOG_DAEMON. If you wish to change it, append its name to the log file name, delimited by a single colon. For example, if *logfile* is the string **syslog:local7** then Amd will log messages via *syslog*(3) using the LOG_LOCAL7 facility (if it exists on the system).

log_options (string, default no logging options)
Same as the **−x** option to Amd. Specify any logging options for Amd. Options are comma delimited, and can be preceded by the string "no" to negate their meaning. The "debug" logging option is only available if am-utils was configured with --enable-debug. You can get the list of supported debugging and logging options by running **amd −H**. Possible values are:

all all messages
debug debug messages
error non-fatal system errors
fatal fatal errors
info information
map map errors
stats additional statistical information
user non-fatal user errors

warn warnings
warning warnings

map_reload_interval (numeric, default=3600)
The number of seconds that Amd will wait before it checks to see if any maps have changed at their source (NIS servers, LDAP servers, files, etc.). Amd will reload only those maps that have changed.

nfs_allow_insecure_port (string, default=no)
Normally Amd will refuse requests coming from unprivileged ports (i.e. ports >= 1024 on Unix systems), so that only privileged users and the kernel can send NFS requests to it. However, some kernels (certain versions of Darwin, MacOS X, and Linux) have bugs that cause them to use unprivileged ports in certain situations, which causes Amd to stop dead in its tracks. This parameter allows Amd to operate normally even on such systems, at the expense of a slight decrease in the security of its operations. If you see messages like "ignoring request from foo:1234, port not reserved" in your Amd log, try enabling this parameter and give it another go.

nfs_proto (string, default to trying version tcp then udp)
By default, Amd tries TCP and then UDP. This option forces the overall NFS protocol used to TCP or UDP. It overrides what is in the Amd maps, and is useful when Amd is compiled with NFSv3 support that may not be stable. With this option you can turn off the complete usage of NFSv3 dynamically (without having to recompile Amd) until such time as NFSv3 support is desired again.

nfs_retransmit_counter (numeric, default=11)
Same as the *retransmit* part of the **−t** *timeout.retransmit* option to Amd. Specifies the number of NFS retransmissions that the kernel will use to communicate with Amd.

nfs_retransmit_counter_udp (numeric, default=11)
Same as the **nfs_retransmit_counter** option, but for all UDP mounts only.

nfs_retransmit_counter_tcp (numeric, default=11)
Same as the **nfs_retransmit_counter** option, but for all TCP mounts only.

nfs_retry_interval (numeric, default=8)
Same as the *timeout* part of the **−t** *timeout.retransmit* option to Amd. Specifies the NFS timeout interval, in *tenths* of seconds, between NFS/RPC retries (for UDP and TCP). This is the value that the kernel will use to communicate with Amd.

Amd relies on the kernel RPC retransmit mechanism to trigger mount retries. The values of the **nfs_retransmit_counter** and the **nfs_retry_interval** parameters change the overall retry interval. Too long an interval gives poor interactive response; too short an interval causes excessive retries.

nfs_retry_interval_udp (numeric, default=8)
Same as the **nfs_retry_interval** option, but for all UDP mounts only.

nfs_retry_interval_tcp (numeric, default=8)
Same as the **nfs_retry_interval** option, but for all TCP mounts only.

nfs_vers (numeric, default to trying version 3 then 2)
By default, Amd tries version 3 and then version 2. This option forces the overall NFS protocol used to version 3 or 2. It overrides what is in the Amd maps, and is useful when Amd is compiled with NFSv3 support that may not be stable. With this option you can turn off the complete usage of NFSv3 dynamically (without having to recompile Amd) until such time as NFSv3 support is desired again.

nis_domain (string, default to local NIS domain name)
Same as the **−y** option to Amd. Specify an alternative NIS domain from which to fetch the NIS maps. The default is the system domain name. This option is ignored if NIS support is not available.

normalize_hostnames (boolean, default=no)
Same as the **−n** option to Amd. If "yes," then the name refereed to by ${rhost} is normalized relative to the host database before being used. The effect is to translate aliases into ‘‘official’’ names.

normalize_slashes (boolean, default=yes)

If "yes," then Amd will condense all multiple ‘‘/’’ (slash) characters into one and remove all trailing slashes. If "no," then Amd will not touch strings that may contain repeated or trailing slashes. The latter is sometimes useful with SMB mounts, which often require multiple slash characters in pathnames.

os (string, default to compiled in value)
Same as the **−O** option to Amd. Allows you to override the compiled-in name of the operating system. Useful when the built-in name is not desired for backward compatibility reasons. For example, if the build in name is ‘‘sunos5’’, you can override it to ‘‘sos5’’, and use older maps which were written with the latter in mind.

osver (string, default to compiled in value)
Same as the **−o** option to Amd. Overrides the compiled-in version number of the operating system. Useful when the built in version is not desired for backward compatibility reasons. For example, if the build in version is ‘‘2.5.1’’, you can override it to ‘‘5.5.1’’, and use older maps that were written with the latter in mind.

pid_file (string, default=/dev/stdout)
Specify a file to store the process ID of the running daemon into. If not specified, Amd will print its process id onto the standard output. Useful for killing Amd after it had run. Note that the PID of a running Amd can also be retrieved via **amq −p**. This file is used only if the print_pid option is on.

plock (boolean, default=yes)
Same as the **−S** option to Amd. If "yes," lock the running executable pages of Amd into memory. To improve Amd’s performance, systems that support the **plock**(3) or **mlockall**(2) call can lock the Amd process into memory. This way there is less chance it the operating system will schedule, page out, and swap the Amd process as needed. This improves Amd’s performance, at the cost of reserving the memory used by the Amd process (making it unavailable for other processes).

portmap_program (numeric, default=300019)
Specify an alternate Port-mapper RPC program number, other than the official number. This is useful when running multiple Amd processes. For example, you can run another Amd in "test" mode, without affecting the primary Amd process in any way. For safety reasons, the alternate program numbers that can be specified must be in the range 300019-300029, inclusive. Amq has an option **-P** which can be used to specify an alternate program number of an Amd to contact. In this way, amq can fully control any number of Amd processes running on the same host.

preferred_amq_port (numeric, default=0)
Specify an alternate Port-mapper RPC port number for Amd's Amq service. This is used for both UDP and TCP. Setting this value to 0 (or not defining it) will cause Amd to select an arbitrary port number. Setting the Amq RPC service port to a specific number is useful in firewalled or NAT'ed environments, where you need to know which port Amd will listen on.

print_pid (boolean, default=no)
Same as the **−p** option to Amd. If "yes," Amd will print its process ID upon starting.

print_version (boolean, default=no)
Same as the **−v** option to Amd, but the version prints and Amd continues to run. If "yes," Amd will print its version information string, which includes some configuration and compilation values.

restart_mounts (boolean, default=no)
Same as the **−r** option to Amd. If "yes" Amd will scan the mount table to determine which file systems are currently mounted. Whenever one of these would have been automounted, Amd inherits it.

show_statfs_entries (boolean), default=no)
If "yes," then all maps which are browsable will also show the number of entries (keys) they have when "df" runs. (This is accomplished by returning non-zero values to the statfs(2) system call).

truncate_log (boolean), default=no)
If "yes," then the log file (if it is a regular file), will be truncated upon startup.

unmount_on_exit (boolean), default=no)
If "yes," then Amd will attempt to unmount all file systems which it knows about. Normally Amd leaves all (esp. NFS) mounted file systems intact. Note that Amd does not know about file systems mounted before it starts up, unless the restart_mounts option or **−r** flag are used.

use_tcpwrappers (boolean), default=yes)
If "yes," then Amd will use the tcpd/librwap tcpwrappers library (if available) to control access to Amd via the /etc/hosts.allow and /etc/hosts.deny files.

vendor (string, default to compiled in value)
The name of the vendor of the operating system. Overrides the compiled-in vendor name. Useful when the compiled-in name is not desired. For example, most Intel based systems set the vendor name to ‘‘unknown’’, but you can set it to ‘‘redhat.’’

Parameters applicable to regular map sections

map_name (string, must be specified)

Name of the map where the keys are located.

tag (string, default no tag)

Each map entry in the configuration file can be tagged. If no tag is specified, that map section will always be processed by Amd. If it is specified, then Amd will process the map if the **-T** option was given to Amd, and the value given to that command-line option matches that in the map section.

EXAMPLES

Here is a real Amd configuration file I use daily.

```
# GLOBAL OPTIONS SECTION
[ global ]
normalize_hostnames =    no
print_pid =              no
restart_mounts =         yes
auto_dir =               /n
log_file =               /var/log/amd
log_options =            all
#debug_options =         all
plock =                  no
selectors_in_defaults =  yes
# config.guess picks up "sunos5" and I don't want to edit my maps yet
os =                     sos5
# if you print_version after setting up "os," it will show it.
print_version =          no
map_type =               file
search_path =            /etc/amdmaps:/usr/lib/amd:/usr/local/AMD/lib
browsable_dirs =         yes

# DEFINE AN AMD MOUNT POINT
[ /u ]
map_name =               amd.u

[ /proj ]
map_name =               amd.proj

[ /src ]
map_name =               amd.src

[ /misc ]
map_name =               amd.misc

[ /import ]
map_name =               amd.import

[ /tftpboot/.amd ]
tag =                    tftpboot
map_name =               amd.tftpboot
```

SEE ALSO

amd(8), **amq**(8), **hosts_access**(5).

‘‘am-utils’’ **info**(1) entry.

Linux NFS and Automounter Administration by Erez Zadok, ISBN 0-7821-2739-8, (Sybex, 2001).

http://www.am-utils.org

Amd − The 4.4 BSD Automounter

AUTHORS

Erez Zadok <ezk@cs.sunysb.edu>, Computer Science Department, Stony Brook University, Stony Brook, New York, USA.

Other authors and contributors to am-utils are listed in the **AUTHORS** file distributed with am-utils.

NAME

ar — a.out archive (library) file format

SYNOPSIS

```
#include <ar.h>
```

DESCRIPTION

The archive command **ar** combines several files into one. Archives are mainly used as libraries of object files intended to be loaded using the link-editor ld(1).

A file created with **ar** begins with the "magic" string "!<arch>\n". The rest of the archive is made up of objects, each of which is composed of a header for a file, a possible file name, and the file contents. The header is portable between machine architectures, and, if the file contents are printable, the archive is itself printable.

The header is made up of six variable length ASCII fields, followed by a two character trailer. The fields are the object name (16 characters), the file last modification time (12 characters), the user and group id's (each 6 characters), the file mode (8 characters) and the file size (10 characters). All numeric fields are in decimal, except for the file mode which is in octal.

The modification time is the file *st_mtime* field, i.e., CUT seconds since the epoch. The user and group id's are the file *st_uid* and *st_gid* fields. The file mode is the file *st_mode* field. The file size is the file *st_size* field. The two-byte trailer is the string "`\n".

Only the name field has any provision for overflow. If any file name is more than 16 characters in length or contains an embedded space, the string "#1/" followed by the ASCII length of the name is written in the name field. The file size (stored in the archive header) is incremented by the length of the name. The name is then written immediately following the archive header.

Any unused characters in any of these fields are written as space characters. If any fields are their particular maximum number of characters in length, there will be no separation between the fields.

Objects in the archive are always an even number of bytes long; files which are an odd number of bytes long are padded with a newline ("\n") character, although the size in the header does not reflect this.

SEE ALSO

ar(1), stat(2)

HISTORY

There have been at least four **ar** formats. The first was denoted by the leading "magic" number 0177555 (stored as type int). These archives were almost certainly created on a 16-bit machine, and contain headers made up of five fields. The fields are the object name (8 characters), the file last modification time (type long), the user id (type char), the file mode (type char) and the file size (type unsigned int). Files were padded to an even number of bytes.

The second was denoted by the leading "magic" number 0177545 (stored as type int). These archives may have been created on either 16 or 32-bit machines, and contain headers made up of six fields. The fields are the object name (14 characters), the file last modification time (type long), the user and group id's (each type char), the file mode (type int), and the file size (type long). Files were padded to an even number of bytes.

Both of these historical formats may be read with ar(1).

The current archive format (without support for long character names and names with embedded spaces) was introduced in 4.0BSD. The headers were the same as the current format, with the exception that names longer than 16 characters were truncated, and names with embedded spaces (and often trailing spaces) were not supported. It has been extended for these reasons, as described above. This format first appeared in

4.4BSD.

COMPATIBILITY

The current a.out archive format is not specified by any standard.

ELF systems use the **ar** format specified by the AT&T System V.4 UNIX ABI, with the same headers but different long file name handling.

BUGS

The <ar.h> header file, and the **ar** manual page, do not currently describe the ELF archive format.

NAME

atf-formats — machine-parseable data formats used by ATF

DESCRIPTION

This manual page describes the multiple data formats used in ATF. These formats affect configuration files, control files and any data that is externalized or internalized by the tools.

Data files are always organized as follows:

```
Header1: Value1           \
    ...                   | head
HeaderN: ValueN           /
                          mandatory blank line
Free-form text contents   \
    ...                   | body
    ...                   /
```

A file must always contain a ‘Content-Type’ header and must always separate that header from the body with a blank line, even if the body is empty.

The ‘Content-Type’ is always of the form:

```
Content-Type: application/X-atf-<subtype>; version="<version>"
```

where ‘subtype’ indicates the specific file format and ‘version’ its format version. This header must be the first one of the file.

The main purpose of the ‘Content-Type’ header, aside from determining the format used in the file, is to allow future changes to a given format. Whenever an incompatible change is made, the version is bumped by one. By keeping the header in the first line, future versions may even remove the need for such a header -- e.g. if some format was replaced by XML files, which have their own mandatory header.

The rest of this document details the different format types.

Format: application/X-atf-atffile, version: 1

Atffiles are logically divided into three sections:

- Test programs: the list of test programs that define the test suite described by the Atffile.
- Meta-data properties: these define some constant values applicable to all the test programs defined in the file. In some sense they define the properties that describe the test suite.
- Configuration variables: defaults for configuration variables that can be overridden through configuration files or the command line.

The grammar for Atffiles is the following:

```
DATA ::= ( ( CONF | PROP | TP )? COMMENT? NEWLINE )* EOF
CONF ::= 'conf:' WORD EQUAL STRING
PROP ::= 'prop:' WORD EQUAL STRING
TP ::= TPFILE | TPGLOB
TPFILE ::= 'tp: ' STRING
TPGLOB ::= 'tp-glob: ' STRING
STRING ::= WORD | '"' WORD* '"'
```

The meaning of the constructions above is:

CONF Definition of a configuration variable.

PROP Definition of a meta-data property.

TPFILE Addition of a test program into the test suite. The string is taken literally as the program's name, and this program must exist.

TPGLOB Addition of multiple test programs into the test suite. The string is taken as a glob pattern, which may have or not have any matches in the current directory.

An example:

```
prop: test-suite = utilities

conf: unprivileged-user = nobody

tp: t_cp
tp: t_mv
tp: t_df
tp-glob: t_dir_*
```

Format: application/X-atf-config, version: 1

Configuration files are very simple: they only contain a list of variable name/variable value pairs. Their grammar is:

```
DATA ::= ( VAR? COMMENT? NEWLINE )* EOF
VAR ::= WORD EQUAL STRING
COMMENT ::= HASH WORD*
STRING ::= WORD | '"' WORD* '"'
```

An example:

```
# This is the system-wide configuration file for ATF.
# The above and this line are comments placed on their own line.

var1 = this is a variable value
var2 = this is another one      # Optional comment at the end.
```

Format: application/X-atf-tcs, version: 1

The 'application/X-atf-tcs' format is used to describe the results of a collection of test cases; in other words, it represents *the output of a test program*. Unfortunately, it is not easy to control, from inside a test program, what it prints to both its standard output and standard error streams. This is specially the case of test programs written in the POSIX shell language, because they are constantly executing external tools that may print unexpected messages at all times. Due to this, ATF imposes no restrictions on what a test program can send to these two channels; in fact, they are encouraged to print as much useful information as possible to aid in the debugging of test failures.

Because we have no control over the two standard streams, the 'application/X-atf-tcs' format describes the structure of a third stream, known as the *results output*, that test programs must generate. (Note that test programs send, by default, the results output to the standard output; use their **-r** flag to change this whenever you need to parse the data.) This stream is decoupled from the other two and has the following grammar:

```
DATA ::= TCS-COUNT TC-STANZA* EOF
TCS-COUNT ::= 'tcs-count' COLON POSITIVE-NUMBER NEWLINE
TC-STANZA ::= TC-START TC-END
TC-START ::= 'tc-start' COLON STRING NEWLINE
TC-END ::= 'tc-end' COLON STRING COMMA TCR NEWLINE
```

```
TCR ::= 'passed' | ('failed' | 'skipped') COMMA STRING
```

The meaning of the constructions above is:

TCS-COUNT Indicates the number of test cases that will be executed. There will be this exact amount of 'TC-STANZA' constructions following it.

TC-START Indicates the beginning of a test case. This is accompanied by the test case's name.

TC-END Indicates the completion of a test case. This is accompanied by the test case's name, its result and the reason associated with this result (if applicable).

There are multiple reasons behind this design:

- The reader of this format must be able to show real-time progress to the user as the test cases are processed. Therefore, the 'TC-START' construction tells the reader *when* a test case has started to process data.
- The reader of this format has to be able to provide useful statistics to the user without having to wait for the end of the file. Hence, the existence of the 'TCS-COUNT' construction located at the beginning of the file.
- Text-based tools have to be able to easily look for the results of a given test case. This is why the 'TC-END' construction duplicate the test case name already provided in 'TC-START'.

An example:

```
tcs-count: 2
tc-start: add
tc-end: add, passed
tc-start: subtract
tc-end: subtract, failed, Calculated an unexpected value
```

Going back to the standard output and standard error streams, the reader has to be able to match the messages in those two streams to the test cases they belong to. To do this, these two streams must print a magic string that separates the output of test cases from each other, which is enough to synchronize their contents with the results output. This string is '__atf_tc_separator__' and it must printed on a line of its own. The last test case should not be followed by this line because the end of file marker takes its role.

Format: application/X-atf-tps, version: 2

The 'application/X-atf-tps' format multiplexes the standard output, standard error and results output streams from multiple test programs into a single data file. This format is used by `atf-run`(1) to report the execution of several test programs and is later parsed by `atf-report`(1) to inform the user of this process. It has the following grammar:

```
DATA ::= INFO* TPS-COUNT TP-STANZA* INFO* EOF
INFO ::= 'info' COLON STRING COMMA STRING NEWLINE
TPS-COUNT ::= 'tps-count' COLON POSITIVE-NUMBER NEWLINE
TP-STANZA ::= TP-START TC-STANZA* TC-END
TP-START ::= 'tp-start' COLON STRING COMMA POSITIVE-NUMBER NEWLINE
TP-END ::= 'tc-end' COLON STRING (COMMA STRING)?
TC-STANZA ::= TC-START (TC-SO | TC-SE)* TC-END
TC-START ::= 'tc-start' COLON STRING NEWLINE
TC-SO ::= 'tc-so' COLON STRING NEWLINE
TC-SE ::= 'tc-se' COLON STRING NEWLINE
TC-END ::= 'tc-end' COLON STRING COMMA TCR NEWLINE
TCR ::= 'passed' | ('failed' | 'skipped') COMMA STRING
```

The meaning of the constructions above is:

TPS-COUNT Indicates the number of test programs that will be executed. There will be this exact amount of ‘TP-STANZA’ constructions following it.

TP-START Indicates the beginning of a test program. This includes the program’s name and the amount of test cases that will follow.

TP-END Indicates the completion of a test program. This is followed by the program’s name and, if the program ended prematurely, an error message indicating the reason of its failure. A successful execution of a test program (regardless of the status of its test cases) must not be accompanied by any reason.

TC-START Indicates the beginning of a test case. This is accompanied by the test case’s name.

TC-SO Contains a text line sent to the standard output stream during the execution of the test case. Leading and trailing space is preserved.

TC-SE Contains a text line sent to the standard error stream during the execution of the test case. Leading and trailing space is preserved.

TC-END Indicates the completion of a test case. This is accompanied by the test case’s name, its result and the reason associated with this result (if applicable).

An example:

```
tps-count: 2
tp-start: calculator, 2
tc-start: add
tc-end: add, passed
tc-start: subtract
tc-so: 3-2 expected to return 1 but got 0
tc-end: subtract, failed, Calculated an unexpected value
tp-end: calculator
tp-start: files, 1
tc-start: copy
tc-se: could not find the cp(1) utility
tc-end: copy, skipped
tp-end: files
```

SEE ALSO

atf(7)

NAME

boot.cfg — configuration file for /boot

DESCRIPTION

The file /boot.cfg is used to alter the behaviour of the standard boot loader described in boot(8). Configuration changes include setting the timeout, choosing a console device, altering the banner text and displaying a menu allowing boot commands to be easily chosen. If a **boot.cfg** file is not present, the system will boot as normal.

FILE FORMAT

The format of the file is a series of lines containing keyword/value pairs separated by an equals sign ('='). There should be no whitespace surrounding the equals sign. Lines beginning with a hash ('#') are comments and will be ignored.

Some keywords can be present multiple times in the file to define additional items. Such keywords are noted below.

banner (may be present multiple times) The text from banner lines is displayed instead of the standard welcome text by the boot loader. Up to 10 lines can be defined. No special character sequences are recognised, so to specify a blank line, a banner line with no value should be given.

menu (may be present multiple times) Used to define a menu item to be displayed to the end-user at boot time which allows a series of boot commands to be run without further typing. The value consists of the required menu text, followed by a colon (':') and then the desired command(s). Multiple commands can be specified separated by a semi-colon. If the specified menu text is empty (the colon appears immediately after the equals sign), then the displayed menu text is the same as the command. For example:

```
menu=Boot normally:boot
menu=Boot single-user:boot -s
menu=Boot with module foo:load /foo.kmod;boot
menu=Boot with serial console:consdev com0;boot
menu=:boot hd1a:netbsd -as
```

Each menu item will be prefixed by an ascending number when displayed, i.e. the order in the **boot.cfg** file is important.

Each command is executed just as though the user had typed it in and so can be any valid command that would be accepted at the normal boot prompt. In addition, 'prompt' can be used to drop to the normal boot prompt.

timeout If the value is greater than zero, this specifies the time in seconds that the boot loader will wait for the end-user to choose a menu item. During the countdown period, they may press Return to choose the default option or press a number key corresponding to a menu option. If any other key is pressed, the countdown will stop and the user will be prompted to choose a menu option with no further time limit. If the timeout value is set to zero, the default option will be booted immediately. If the timeout value is negative or is not a number, there will be no time limit for the user to choose an option.

default Used to specify the default menu item which will be chosen in the case of Return being pressed or the timeout timer reaching zero. The value is the number of the menu item as displayed. As described above, the menu items are counted from 1 in the order listed in **boot.cfg**. If not specified, the default value will be option 1, i.e. the first item.

consdev Changes the console device to that specified in the value. Valid values are any of those that could be specified at the normal boot prompt with the consdev command.

load Used to load kernel modules, which will be passed on to the kernel for initialization during early boot. The argument is the complete path and file name of the module to be loaded. May be used as many times as needed.

EXAMPLES

Here is an example **boot.cfg** file:

```
banner=Welcome to NetBSD
banner=================
banner=
banner=Please choose an option from the following menu:
menu=Boot normally:boot
menu=Boot single-user:boot -s
menu=Boot from second disk:boot hd1a:
menu=Boot with module foo:load /foo.kmod;boot
menu=Boot with modules foo and bar:load /foo.kmod;load /bar.kmod;boot
menu=Go to command line (advanced users only):prompt
timeout=-1
default=1
# Always load ramdisk module
load=/ramdisk.kmod
```

This will display:

```
Welcome to NetBSD
=================

Please choose an option from the following menu:

      1. Boot normally
      2. Boot single-user
      3. Boot from second disk
      4. Boot with module foo
      5. Boot with modules foo and bar
      6. Go to command line (advanced users only)

Option [1]:
```

It will then wait for the user to type 1, 2, 3, 4, 5 or 6 followed by Return. Pressing Return by itself will run option 1. There will be no timeout.

SEE ALSO

boot(8)

BUGS

Support for **boot.cfg** is currently for NetBSD/i386 and NetBSD/amd64 only. It is hoped that its use will be extended to other appropriate ports that use the boot(8) interface.

HISTORY

The **boot.cfg** utility appeared in NetBSD 5.0.

AUTHORS

The **boot.cfg** extensions to boot(8) were written by Stephen Borrill ⟨sborrill@NetBSD.org⟩.

NAME

bootparams — boot parameter database

SYNOPSIS

/etc/bootparams

DESCRIPTION

The **bootparams** file specifies the boot parameters that diskless(8) clients may request when booting over the network. Each client supported by this server must have an entry in the **bootparams** file containing the servers and pathnames for its root, area. It may optionally contain swap, dump, and other entry types.

Each line in the file (other than comment lines that begin with a #) specifies the client name followed by the pathnames that the client may request by their logical names. Names are matched in a case-insensitive fashion, and may also be wildcarded using shell-style globbing characters.

The components of the line are delimited with blank or tab, and may be continued onto multiple lines with a backslash.

For example:

```
dummy   root=server:/export/dummy/root \
        swap=server:/export/dummy/swap \
        dump=server:/export/dummy/swap \
        gateway=router:255.255.255.0
```

When the client named "dummy" requests the pathname for its logical "root" it will be given server "server" and pathname "/export/dummy/root" as the response to its RPC request.

```
netra[1-5]www* root=server:/export/jumpstart/netra_www
```

When any client with a name matching the pattern "netra[1-5]www*" requests the pathname for its logical "root" it will be given server "server" and pathname "/export/jumpstart/netra_www" as the response to its RPC request. As this example implies, this is useful for setting up Jumpstart servers for Sun clients.

NOTES

The server does not default to the localhost, and must be filled in.

FILES

/etc/bootparams default configuration file

SEE ALSO

diskless(8), rpc.bootparamd(8)

NAME

bootptab − Internet Bootstrap Protocol server database

DESCRIPTION

The *bootptab* file is the configuration database file for *bootpd*, the Internet Bootstrap Protocol server. Its format is similar to that of *termcap*(5) in which two-character case-sensitive tag symbols are used to represent host parameters. These parameter declarations are separated by colons (:), with a general format of:

hostname:tg=value:tg=value:tg=value:

where *hostname* is the actual name of a bootp client (or a "dummy entry"), and *tg* is a two-character tag symbol. Replies are returned to clients only if an entry with the client's Ethernet or IP address exists in the *booptab* file. Dummy entries have an invalid hostname (one with a "." as the first character) and are used to provide default values used by other entries via the **tc=.dummy-entry** mechanism. Most tags must be followed by an equal sign and a value as above. Some may also appear in a boolean form with no value (i.e. :*tg*:). The currently recognized tags are:

bf	Bootfile
bs	Bootfile size in 512-octet blocks
cs	Cookie server address list
df	Merit dump file
dn	Domain name
ds	Domain name server address list
ef	Extension file
gw	Gateway address list
ha	Host hardware address
hd	Bootfile home directory
hn	Send client's hostname to client
ht	Host hardware type (see Assigned Numbers RFC)
im	Impress server address list
ip	Host IP address
lg	Log server address list
lp	LPR server address list
ns	IEN-116 name server address list
nt	NTP (time) Server (RFC 1129)
ra	Reply address override
rl	Resource location protocol server address list
rp	Root path to mount as root
sa	TFTP server address client should use
sm	Host subnet mask
sw	Swap server address
tc	Table continuation (points to similar "template" host entry)
td	TFTP root directory used by "secure" TFTP servers
to	Time offset in seconds from UTC
ts	Time server address list
vm	Vendor magic cookie selector
yd	YP (NIS) domain name
ys	YP (NIS) server address

There is also a generic tag, T*n*, where *n* is an RFC1084 vendor field tag number. Thus it is possible to immediately take advantage of future extensions to RFC1084 without being forced to modify *bootpd* first. Generic data may be represented as either a stream of hexadecimal numbers or as a quoted string of ASCII characters. The length of the generic data is automatically determined and inserted into the proper field(s) of the RFC1084-style bootp reply.

The following tags take a whitespace-separated list of IP addresses: **cs**, **ds**, **gw**, **im**, **lg**, **lp**, **ns**, **nt**, **ra**, **rl**, and **ts**. The **ip**, **sa**, **sw**, **sm**, and **ys** tags each take a single IP address. All IP addresses are specified in standard

Internet "dot" notation and may use decimal, octal, or hexadecimal numbers (octal numbers begin with 0, hexadecimal numbers begin with '0x' or '0X'). Any IP addresses may alternatively be specified as a hostname, causing *bootpd* to lookup the IP address for that host name using gethostbyname(3). If the **ip** tag is not specified, *bootpd* will determine the IP address using the entry name as the host name. (Dummy entries use an invalid host name to avoid automatic IP lookup.)

The **ht** tag specifies the hardware type code as either an unsigned decimal, octal, or hexadecimal integer or one of the following symbolic names: **ethernet** or **ether** for 10Mb Ethernet, **ethernet3** or **ether3** for 3Mb experimental Ethernet, **ieee802**, **tr**, or **token-ring** for IEEE 802 networks, **pronet** for Proteon ProNET Token Ring, or **chaos**, **arcnet**, or **ax.25** for Chaos, ARCNET, and AX.25 Amateur Radio networks, respectively. The **ha** tag takes a hardware address which may be specified as a host name or in numeric form. Note that the numeric form *must* be specified in hexadecimal; optional periods and/or a leading '0x' may be included for readability. The **ha** tag must be preceded by the **ht** tag (either explicitly or implicitly; see **tc** below). If the hardware address is not specified and the type is specified as either "ethernet" or "ieee802", then *bootpd* will try to determine the hardware address using ether_hostton(3).

The hostname, home directory, and bootfile are ASCII strings which may be optionally surrounded by double quotes ("). The client's request and the values of the **hd** and **bf** symbols determine how the server fills in the bootfile field of the bootp reply packet.

If the **bf** option is specified, its value is copied into the reply packet. Otherwise, the name supplied in the client request is used. If the **hd** option is specified, its value is prepended to the boot file in the reply packet, otherwise the path supplied in the client request is used. The existence of the boot file is NOT verified by *bootpd* because the boot file may be on some other machine.

The **bs** option specified the size of the boot file. It can be written as **bs**=auto which causes *bootpd* to determine the boot file size automatically.

Some newer versions of *tftpd* provide a security feature to change their root directory using the *chroot*(2) system call. The **td** tag may be used to inform *bootpd* of this special root directory used by *tftpd*. (One may alternatively use the *bootpd* "-c chdir" option.) The **hd** tag is actually relative to the root directory specified by the **td** tag. For example, if the real absolute path to your BOOTP client bootfile is /tftpboot/bootfiles/bootimage, and *tftpd* uses /tftpboot as its "secure" directory, then specify the following in *bootptab*:

```
:td=/tftpboot:hd=/bootfiles:bf=bootimage:
```

If your bootfiles are located directly in /tftpboot, use:

```
:td=/tftpboot:hd=/:bf=bootimage:
```

The **sa** tag may be used to specify the IP address of the particular TFTP server you wish the client to use. In the absence of this tag, *bootpd* will tell the client to perform TFTP to the same machine *bootpd* is running on.

The time offset **to** may be either a signed decimal integer specifying the client's time zone offset in seconds from UTC, or the keyword **auto** which uses the server's time zone offset. Specifying the **to** symbol as a boolean has the same effect as specifying **auto** as its value.

The bootfile size **bs** may be either a decimal, octal, or hexadecimal integer specifying the size of the bootfile in 512-octet blocks, or the keyword **auto** which causes the server to automatically calculate the bootfile size at each request. As with the time offset, specifying the **bs** symbol as a boolean has the same effect as specifying **auto** as its value.

The vendor magic cookie selector (the **vm** tag) may take one of the following keywords: **auto** (indicating that vendor information is determined by the client's request), **rfc1048** or **rfc1084** (which always forces an RFC1084-style reply), or **cmu** (which always forces a CMU-style reply).

The **hn** tag is strictly a boolean tag; it does not take the usual equals-sign and value. It's presence indicates that the hostname should be sent to RFC1084 clients. *Bootpd* attempts to send the entire hostname as it is specified in the configuration file; if this will not fit into the reply packet, the name is shortened to just the host field (up to the first period, if present) and then tried. In no case is an arbitrarily-truncated hostname

sent (if nothing reasonable will fit, nothing is sent).

Often, many host entries share common values for certain tags (such as name servers, etc.). Rather than repeatedly specifying these tags, a full specification can be listed for one host entry and shared by others via the **tc** (table continuation) mechanism. Often, the template entry is a dummy host which doesn't actually exist and never sends bootp requests. This feature is similar to the **tc** feature of *termcap*(5) for similar terminals. Note that *bootpd* allows the **tc** tag symbol to appear anywhere in the host entry, unlike *termcap* which requires it to be the last tag. Information explicitly specified for a host always overrides information implied by a **tc** tag symbol, regardless of its location within the entry. The value of the **tc** tag may be the hostname or IP address of any host entry previously listed in the configuration file.

Sometimes it is necessary to delete a specific tag after it has been inferred via **tc**. This can be done using the construction *tag***@** which removes the effect of *tag* as in *termcap*(5). For example, to completely undo an IEN-116 name server specification, use ":ns@:" at an appropriate place in the configuration entry. After removal with **@**, a tag is eligible to be set again through the **tc** mechanism.

Blank lines and lines beginning with "#" are ignored in the configuration file. Host entries are separated from one another by newlines; a single host entry may be extended over multiple lines if the lines end with a backslash (\). It is also acceptable for lines to be longer than 80 characters. Tags may appear in any order, with the following exceptions: the hostname must be the very first field in an entry, and the hardware type must precede the hardware address.

An example */etc/bootptab* file follows:

```
# Sample bootptab file (domain=andrew.cmu.edu)

.default:\
        :hd=/usr/boot:bf=null:\
        :ds=netserver, lancaster:\
        :ns=pcs2, pcs1:\
        :ts=pcs2, pcs1:\
        :sm=255.255.255.0:\
        :gw=gw.cs.cmu.edu:\
        :hn:to=-18000:

carnegie:ht=6:ha=7FF8100000AF:tc=.default:
baldwin:ht=1:ha=0800200159C3:tc=.default:
wylie:ht=1:ha=00DD00CADF00:tc=.default:
arnold:ht=1:ha=0800200102AD:tc=.default:
bairdford:ht=1:ha=08002B02A2F9:tc=.default:
bakerstown:ht=1:ha=08002B0287C8:tc=.default:

# Special domain name server and option tags for next host
butlerjct:ha=08002001560D:ds=128.2.13.42:\
        :T37=0x12345927AD3BCF:\
        :T99="Special ASCII string":\
        :tc=.default:

gastonville:ht=6:ha=7FFF81000A47:tc=.default:
hahntown:ht=6:ha=7FFF81000434:tc=.default:
hickman:ht=6:ha=7FFF810001BA:tc=.default:
lowber:ht=1:ha=00DD00CAF000:tc=.default:
mtoliver:ht=1:ha=00DD00FE1600:tc=.default:
```

FILES

/etc/bootptab

SEE ALSO

bootpd(8), tftpd(8),
DARPA Internet Request For Comments RFC951, RFC1048, RFC1084, Assigned Numbers

NAME

bounce − Postfix bounce message template format

SYNOPSIS

bounce_template_file = /etc/postfix/bounce.cf

postconf -b [*template_file*]

DESCRIPTION

The Postfix **bounce**(8) server produces delivery status notification (DSN) messages for undeliverable mail, delayed mail, successful delivery or address verification requests.

By default, these notifications are generated from built-in templates with message headers and message text. Sites can override the built-in information by specifying a bounce template file with the **bounce_template_file** configuration parameter.

This document describes the general procedure to create a bounce template file, followed by the specific details of bounce template formats.

GENERAL PROCEDURE

To create a customized bounce template file, create a temporary copy of the file **/etc/postfix/bounce.cf.default** and edit the temporary file.

To preview the results of $*name* expansions in the template text, use the command

postconf -b *temporary_file*

Errors in the template will be reported to the standard error stream and to the syslog daemon.

While previewing the text, be sure to pay particular attention to the expansion of time value parameters that appear in the delayed mail notification text.

Once the result is satisfactory, copy the template to the Postfix configuration directory and specify in main.cf something like:

/etc/postfix/main.cf:
bounce_template_file = /etc/postfix/bounce.cf

TEMPLATE FILE FORMAT

The template file can specify templates for failed mail, delayed mail, successful delivery or for address verification. These templates are named **failure_template**, **delay_template**, **success_template** and **verify_template**, respectively. You can but do not have to specify all four templates in a bounce template file.

Each template starts with "*template_name* **= <<EOF**" and ends with a line that contains the word "**EOF**" only. You can change the word EOF, but you can't enclose it in quotes as with the shell or with Perl (*template_name* **= <<'EOF'**). Here is an example:

```
# The failure template is used for undeliverable mail.

failure_template = <<EOF
Charset: us-ascii
From: MAILER-DAEMON (Mail Delivery System)
Subject: Undelivered Mail Returned to Sender
Postmaster-Subject: Postmaster Copy: Undelivered Mail

This is the mail system at host $myhostname.
```

I'm sorry to have to inform you that your message could not
be delivered to one or more recipients. It's attached below.

For further assistance, please send mail to postmaster.

If you do so, please include this problem report. You can
delete your own text from the attached returned message.

The mail system
EOF

The usage and specification of bounce templates is subject to the following restrictions:

- No special meaning is given to the backslash character or to leading whitespace; these are always taken literally.
- Inside the << context, the "$" character is special. To produce a "$" character as output, specify "$$".
- Outside the << context, lines beginning with "#" are ignored, as are empty lines, and lines consisting of whitespace only.

Examples of all templates can be found in the file **bounce.cf.default** in the Postfix configuration directory.

TEMPLATE HEADER FORMAT

The first portion of a bounce template consists of optional template headers. These either become message headers in the delivery status notification, or control the formatting of the notification. Headers not specified in a template will be left at their default value.

The following headers are supported:

Charset:
The MIME character set of the template message text. See the "TEMPLATE MESSAGE TEXT FORMAT" description below.

From: The sender address in the message header of the delivery status notification.

Subject:
The subject in the message header of the delivery status notification.

Postmaster-Subject:
The subject that will be used in Postmaster copies of undeliverable or delayed mail notifications. These copies are sent under control of the notify_classes configuration parameter.

The usage and specification of template message headers is subject to the following restrictions:

- Template message header names can be specified in upper case, lower case or mixed case. Postfix always uses the spelling as shown in the example above.
- Template message headers must not span multiple lines.
- Template message headers must not contain main.cf $parameters.
- Template message headers must contain ASCII characters only.

TEMPLATE MESSAGE TEXT FORMAT

The second portion of a bounce template consists of message text. As the above example shows, template message text may contain main.cf $parameters. Besides the parameters that are defined in main.cf, the following parameters are treated specially depending on the suffix that is appended to their name.

delay_warning_time_*suffix*
Expands into the value of the **delay_warning_time** parameter, expressed in the time unit specified by *suffix*, which is one of **seconds**, **minutes**, **hours, days**, or **weeks**.

maximal_queue_lifetime_*suffix*

Expands into the value of the **maximal_queue_lifetime** parameter, expressed in the time unit specified by *suffix*. See above under **delay_warning_time** for possible *suffix* values.

The usage and specification of template message text is subject to the following restrictions:

- The template message text is not sent in Postmaster copies of delivery status notifications.
- If the template message text contains non-ASCII characters, Postfix requires that the **Charset:** template header is updated. Specify an appropriate superset of US-ASCII. A superset is needed because Postfix appends ASCII text after the message template when it sends a delivery status notification.

SEE ALSO

bounce(8), Postfix delivery status notifications
postconf(5), configuration parameters

LICENSE

The Secure Mailer license must be distributed with this software.

HISTORY

The Postfix bounce template format was originally developed by Nicolas Riendeau.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

canonical − Postfix canonical table format

SYNOPSIS

postmap /etc/postfix/canonical

postmap -q "*string***" /etc/postfix/canonical**

postmap -q - /etc/postfix/canonical <*inputfile*

DESCRIPTION

The optional **canonical**(5) table specifies an address mapping for local and non-local addresses. The mapping is used by the **cleanup**(8) daemon, before mail is stored into the queue. The address mapping is recursive.

Normally, the **canonical**(5) table is specified as a text file that serves as input to the **postmap**(1) command. The result, an indexed file in **dbm** or **db** format, is used for fast searching by the mail system. Execute the command "**postmap /etc/postfix/canonical**" to rebuild an indexed file after changing the corresponding text file.

When the table is provided via other means such as NIS, LDAP or SQL, the same lookups are done as for ordinary indexed files.

Alternatively, the table can be provided as a regular-expression map where patterns are given as regular expressions, or lookups can be directed to TCP-based server. In those cases, the lookups are done in a slightly different way as described below under "REGULAR EXPRESSION TABLES" or "TCP-BASED TABLES".

By default the **canonical**(5) mapping affects both message header addresses (i.e. addresses that appear inside messages) and message envelope addresses (for example, the addresses that are used in SMTP protocol commands). This is controlled with the **canonical_classes** parameter.

NOTE: Postfix versions 2.2 and later rewrite message headers from remote SMTP clients only if the client matches the local_header_rewrite_clients parameter, or if the remote_header_rewrite_domain configuration parameter specifies a non-empty value. To get the behavior before Postfix 2.2, specify "local_header_rewrite_clients = static:all".

Typically, one would use the **canonical**(5) table to replace login names by *Firstname.Lastname*, or to clean up addresses produced by legacy mail systems.

The **canonical**(5) mapping is not to be confused with *virtual alias* support or with local aliasing. To change the destination but not the headers, use the **virtual**(5) or **aliases**(5) map instead.

CASE FOLDING

The search string is folded to lowercase before database lookup. As of Postfix 2.3, the search string is not case folded with database types such as regexp: or pcre: whose lookup fields can match both upper and lower case.

TABLE FORMAT

The input format for the **postmap**(1) command is as follows:

pattern result
: When *pattern* matches a mail address, replace it by the corresponding *result*.

blank lines and comments
: Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

multi-line text

A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

TABLE SEARCH ORDER

With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, patterns are tried in the order as listed below:

user@domain address

Replace *user@domain* by *address*. This form has the highest precedence.

This is useful to clean up addresses produced by legacy mail systems. It can also be used to produce *Firstname.Lastname* style addresses, but see below for a simpler solution.

user address

Replace *user@site* by *address* when *site* is equal to $**myorigin**, when *site* is listed in $**mydestination**, or when it is listed in $**inet_interfaces** or $**proxy_interfaces**.

This form is useful for replacing login names by *Firstname.Lastname*.

@domain address

Replace other addresses in *domain* by *address*. This form has the lowest precedence.

Note: *@domain* is a wild-card. When this form is applied to recipient addresses, the Postfix SMTP server accepts mail for any recipient in *domain*, regardless of whether that recipient exists. This may turn your mail system into a backscatter source: Postfix first accepts mail for non-existent recipients and then tries to return that mail as "undeliverable" to the often forged sender address.

RESULT ADDRESS REWRITING

The lookup result is subject to address rewriting:

- When the result has the form *@otherdomain*, the result becomes the same *user* in *otherdomain*.
- When "**append_at_myorigin=yes**", append "**@$myorigin**" to addresses without "@domain".
- When "**append_dot_mydomain=yes**", append "**.$mydomain**" to addresses without ".domain".

ADDRESS EXTENSION

When a mail address localpart contains the optional recipient delimiter (e.g., *user+foo@domain*), the lookup order becomes: *user+foo@domain*, *user@domain*, *user+foo*, *user*, and *@domain*.

The **propagate_unmatched_extensions** parameter controls whether an unmatched address extension (*+foo*) is propagated to the result of table lookup.

REGULAR EXPRESSION TABLES

This section describes how the table lookups change when the table is given in the form of regular expressions. For a description of regular expression lookup table syntax, see **regexp_table**(5) or **pcre_table**(5).

Each pattern is a regular expression that is applied to the entire address being looked up. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Patterns are applied in the order as specified in the table, until a pattern is found that matches the search string.

Results are the same as with indexed file lookups, with the additional feature that parenthesized substrings from the pattern can be interpolated as **$1**, **$2** and so on.

TCP-BASED TABLES

This section describes how the table lookups change when lookups are directed to a TCP-based server. For a description of the TCP client/server lookup protocol, see **tcp_table**(5). This feature is not available up to

and including Postfix version 2.4.

Each lookup operation uses the entire address once. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Results are the same as with indexed file lookups.

BUGS

The table format does not understand quoting conventions.

CONFIGURATION PARAMETERS

The following **main.cf** parameters are especially relevant. The text below provides only a parameter summary. See **postconf**(5) for more details including examples.

canonical_classes
What addresses are subject to canonical address mapping.

canonical_maps
List of canonical mapping tables.

recipient_canonical_maps
Address mapping lookup table for envelope and header recipient addresses.

sender_canonical_maps
Address mapping lookup table for envelope and header sender addresses.

propagate_unmatched_extensions
A list of address rewriting or forwarding mechanisms that propagate an address extension from the original address to the result. Specify zero or more of **canonical**, **virtual**, **alias**, **forward**, **include**, or **generic**.

Other parameters of interest:

inet_interfaces
The network interface addresses that this system receives mail on. You need to stop and start Postfix when this parameter changes.

local_header_rewrite_clients
Rewrite message header addresses in mail from these clients and update incomplete addresses with the domain name in $myorigin or $mydomain; either don't rewrite message headers from other clients at all, or rewrite message headers and update incomplete addresses with the domain specified in the remote_header_rewrite_domain parameter.

proxy_interfaces
Other interfaces that this machine receives mail on by way of a proxy agent or network address translator.

masquerade_classes
List of address classes subject to masquerading: zero or more of **envelope_sender**, **envelope_recipient**, **header_sender**, **header_recipient**.

masquerade_domains
List of domains that hide their subdomain structure.

masquerade_exceptions
List of user names that are not subject to address masquerading.

mydestination
List of domains that this mail system considers local.

myorigin
The domain that is appended to locally-posted mail.

owner_request_special
Give special treatment to **owner-***xxx* and *xxx*-**request** addresses.

remote_header_rewrite_domain
Don’t rewrite message headers from remote clients at all when this parameter is empty; otherwise, rewrite message headers and append the specified domain name to incomplete addresses.

SEE ALSO

cleanup(8), canonicalize and enqueue mail
postmap(1), Postfix lookup table manager
postconf(5), configuration parameters
virtual(5), virtual aliasing

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview
ADDRESS_REWRITING_README, address rewriting guide

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

ccd.conf — concatenated disk driver configuration file

SYNOPSIS

ccd.conf

DESCRIPTION

The **ccd.conf** file defines the configuration of concatenated disk devices, or ccds. It is used by ccdconfig(8) when invoked with the **-C** option and at system boot time. For more information about the concatenated disk driver, see ccd(4).

Each line of the configuration file contains arguments as per the **-c** argument to ccdconfig(8):

ccd ileave [*flags*] *dev* [*. . .*]

ccd The name of the ccd.

ileave The interleave, expressed in units of DEV_BSIZE.

flags The flags for the device, which may be represented as a decimal number, a hexadecimal number, a comma-separated list of strings, or the word “none”.

dev [*. . .*]
The component partitions to be concatenated, which should be of type FS_CCD.

A ‘#’ is a comment, and everything to end of line is ignored. A ‘\’ at the end of a line indicates that the next line should be concatenated with the current. A ‘\’ preceding any character (other than the end of line) prevents that character’s special meaning from taking effect.

FILES

/etc/ccd.conf

EXAMPLES

An example /etc/ccd.conf:

```
#
# /etc/ccd.conf
# Configuration file for concatenated disk devices
#

# ccd           ileave  flags   component devices
ccd0            16      none    /dev/sd2e /dev/sd3e
```

SEE ALSO

ccd(4), ccdconfig(8)

HISTORY

The **ccd.conf** configuration file first appeared in NetBSD 1.1.

NAME

cidr_table − format of Postfix CIDR tables

SYNOPSIS

postmap -q "*string***" cidr:/etc/postfix/***filename*

postmap -q - cidr:/etc/postfix/*filename* <*inputfile*

DESCRIPTION

The Postfix mail system uses optional lookup tables. These tables are usually in **dbm** or **db** format. Alternatively, lookup tables can be specified in CIDR (Classless Inter-Domain Routing) form. In this case, each input is compared against a list of patterns. When a match is found, the corresponding result is returned and the search is terminated.

To find out what types of lookup tables your Postfix system supports use the "**postconf -m**" command.

To test lookup tables, use the "**postmap -q**" command as described in the SYNOPSIS above.

TABLE FORMAT

The general form of a Postfix CIDR table is:

network_address/*network_mask* *result*

When a search string matches the specified network block, use the corresponding *result* value. Specify 0.0.0.0/0 to match every IPv4 address, and ::/0 to match every IPv6 address.

An IPv4 network address is a sequence of four decimal octets separated by ".", and an IPv6 network address is a sequence of three to eight hexadecimal octet pairs separated by ":".

Before comparisons are made, lookup keys and table entries are converted from string to binary. Therefore table entries will be matched regardless of redundant zero characters.

Note: address information may be enclosed inside "[]" but this form is not required.

IPv6 support is available in Postfix 2.2 and later.

network_address *result*

When a search string matches the specified network address, use the corresponding *result* value.

blank lines and comments

Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

multi-line text

A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

TABLE SEARCH ORDER

Patterns are applied in the order as specified in the table, until a pattern is found that matches the search string.

EXAMPLE SMTPD ACCESS MAP

/etc/postfix/main.cf:

```
smtpd_client_restrictions = ... cidr:/etc/postfix/client.cidr ...
```

/etc/postfix/client.cidr:

```
# Rule order matters. Put more specific whitelist entries
# before more general blacklist entries.
192.168.1.1          OK
192.168.0.0/16       REJECT
```

SEE ALSO

postmap(1), Postfix lookup table manager
regexp_table(5), format of regular expression tables
pcre_table(5), format of PCRE tables

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview

AUTHOR(S)

The CIDR table lookup code was originally written by:
Jozsef Kadlecsik
KFKI Research Institute for Particle and Nuclear Physics
POB. 49
1525 Budapest, Hungary

Adopted and adapted by:
Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

config — kernel configuration file syntax

DESCRIPTION

The kernel configuration file specifies the way the kernel should be compiled by the rest of the toolchain. It is processed by config(1) to produce a number of files that will allow the user to compile a possibly customised kernel. One compilation can issue several kernel binaries, with different root and dump devices configurations, or with full debugging information.

This manual page is intended to serve as a complete reference of all aspects of the syntax used in the many files processed by config(1). The novice user will prefer looking at the examples given in config.samples(5) in order to understand better how the default configuration can be changed, and how all of its elements interact with each other.

The kernel configuration file actually contains the description of all the options, drivers and source files involved in the kernel compilation, and the logic that binds them. The **machine** statement, usually found in the std.${MACHINE} file, hides this from the user by automatically including all the descriptive files spread all around the kernel source tree, the main one being conf/files.

Thus, the kernel configuration file contains two parts: the description of the compilation options, and the selection of those options. However, it begins with a small preamble that controls a couple of options of config(1), and a few statements belong to any of the two sections.

The user controls the options selection part, which is located in a file commonly referenced as the *main configuration file* or simply the *kernel configuration file*. The developer is responsible for describing the options in the relevant files from the kernel source tree.

Statements are separated by new-line characters. However, new-line characters can appear in the middle of a given statement, with the value of a space character.

OBJECTS AND NAMES

config(1) is a rather complicated piece of software that tries to comply with any configuration the user might think of. Quite a few different objects are manipulated through the kernel configuration file, therefore some definitions are needed.

Options and attributes

The basic objects driving the kernel compilation are *options*, and are called *attributes* in some contexts. An *attribute* usually refers to a feature a given piece of hardware might have. However, the scope of an attribute is rather wide and can just be a place holder to group some source files together.

There is a special class of attribute, named *interface attribute*, which represents a hook that allows a device to attach to (i.e., be a child of) another device. An *interface attribute* has a (possibly empty) list of *locators* to match the actual location of a device. For example, on a PCI bus, devices are located by a *device number* that is fixed by the wiring of the motherboard. Additionally, each of those devices can appear through several interfaces named *functions*. A single PCI device entity is a unique function number of a given device from the considered PCI bus. Therefore, the locators for a pci(4) device are *dev* (for device), and *function*.

A *locator* can either be a single integer value, or an array of integer values. It can have a default value, in which case it can be wildcarded with a "?" in the options selection section of the configuration file. A single *locator* definition can take one of the following forms:

1. *locator*
2. *locator* = *value*

3. *locator*[*length*]
4. *locator*[*length*] = {*value*, ...}

The variants that specify a default value can be enclosed into square brackets, in which case the locator will not have to be specified later in the options selection section of the configuration file.

In the options selection section, the locators are specified when declaring an instance as a space-separated list of "⟨*locator*⟩ ⟨*value*⟩" where value can be the "?" wildcard if the locator allows it.

Devices, instances and attachments

The main benefit of the kernel configuration file is to allow the user to avoid compiling some drivers, and wire down the configuration of some others. We have already seen that devices attach to each other through *interface attributes*, but not everything can attach to anything. Furthermore, the user has the ability to define precise instances for the devices. An *instance* is simply the reality of a device when it is probed and attached by the kernel.

Each driver has a name for its devices. It is called the base device name and is found as *base* in this documentation. An *instance* is the concatenation of a device name and a number. In the kernel configuration file, instances can sometimes be wildcarded (i.e., the number is replaced by a "∗" or a "?") in order to match all the possible instances of a device.

The usual "∗" becomes a "?" when the instance name is used as an *attachment name*. In the options selection part of the kernel configuration files, an *attachment* is an *interface attribute* concatenated with a number or the wildcard "?".

Pseudo-devices

Some components of the kernel behave like a device although they don't have any actual reality in the hardware. For example, this is the case for special network devices, such as `tun`(4) and `tap`(4). They are integrated in the kernel as pseudo-devices, and can have several instances and even children, just like normal devices.

Dependencies

The options description part of the kernel configuration file contains all the logic that ties the source files together, and it is done first through writing down dependencies between `config`(1) objects.

In this documentation, the syntax for *dependencies* is a comma-separated list of *options* and *attributes*.

For example, the use of an Ethernet network card requires the source files that handle the specificities of that protocol. Therefore, all Ethernet network card drivers depend on the *ether* attribute.

Conditions

Finally, source file selection is possible through the help of conditionals, referred to as *condition* later in this documentation. The syntax for those conditions uses well-known operators ("&", "|" and "!") to combine *options* and *attributes*.

CONTEXT NEUTRAL STATEMENTS

`version` *yyyymmdd*

Indicates the syntax version used by the rest of the file, or until the next **`version`** statement. The argument is an ISO date. A given `config`(1) binary might only be compatible with a limited range of version numbers.

`include` *path*

Includes a file. The path is relative to the top of the kernel source tree, or the inner-most defined **`prefix`**.

cinclude *path*
Conditionally includes a file. Contrary to **include**, it will not produce an error if the file does not exist. The argument obeys the same rules as for **include**.

prefix [*path*]
If *path* is given, it pushes a new prefix for **include** and **cinclude**. **prefix** statements act like a stack, and an empty *path* argument has the latest prefix popped out. The *path* argument is either absolute or relative to the current defined prefix, which defaults to the top of ther kernel source tree.

ifdef *attribute*

ifndef *attribute*

elifdef *attribute*

elifndef *attribute*

else

endif
Conditionally interprets portions of the current file. Those statements depend on whether or not the given *attribute* has been previously defined, through **define** or any other statement that implicitely defines attributes such as **device**.

PREAMBLE

In addition to **include**, **cinclude**, and **prefix**, the preamble may contain the following optional statements:

build *path*
Defines the build directory for the compilation of the kernel. It replaces the default of `../compile/<config-file>` and is superseded by the **-b** parameter of `config`(1).

source *path*
Defines the directory in which the source of the kernel lives. It replaces the default of `../../../..` and is superseded by the **-s** parameter of `config`(1).

OPTIONS DESCRIPTION

The user will not usually have to use descriptive statements, as they are meant for the developer to tie a given piece of code to the rest of the kernel. However, third parties may provide sources to add to the kernel compilation, and the logic that binds them to the NetBSD kernel will have to be added to the user-edited configuration file.

devclass *class*
Defines a special attribute, named *device class*. A given device cannot belong to more than one device class. `config`(1) translates that property by the rule that a device cannot depend on more than one device class, and will properly fill the configuration information file it generates according to that value.

defflag [*file*] *option* [*option* [*...*]] [: *dependencies*]
Defines a boolean option, that can either be selected or be un-selected by the user with the **options** statement. The optional *file* argument names a header file that will contain the C pre-processor definition for the option. If no file name is given, it will default to *opt_<option>.h*. `config`(1) will always create the header file, but if the user choose not to select the option, it will be empty. Several options can be combined in one header file, for convenience. The header file is created in the compilation directory, making them directly accessible by source files.

defparam [*file*] *option* [= *value*] [:= *lint-value*] [*option* [*...*]] [: *dependencies*]
Behaves like **defflag**, except the defined option must have a value. Such options are not typed: they can have either a numeric or a string value. If a *value* is specified, it is treated as a default, and the option is

always defined in the corresponding header file. If a *lint-value* is specified, config(1) will use it as a value when generating a lint configuration with **-L**, and ignore it in all other cases.

deffs [*file*] *name* [*name* [. . .]]
Defines a file-system name. It is no more than a regular option, as defined by **defflag**, but it allows the user to select the file-systems to be compiled in the kernel with the **file-system** statement instead of the **options** statement, and config(1) will enforce the rule that the user must select at least one file-system.

obsolete defflag [*file*] *option* [*option* [. . .]]

obsolete defparam [*file*] *option* [*option* [. . .]]
Those two statements are identical and mark the listed option names as obsolete. If the user selects one of the listed options in the kernel configuration file, config(1) will emit a warning and ignore the option. The optional *file* argument should match the original definition of the option.

define *attribute* [{*locators*}] [: *dependencies*]
Defines an *attribute*. The *locators* list is optional, and can be empty. If the pair of brackets are present, the locator list is defined and the declared attribute becomes an *interface attribute*, on which devices can attach.

maxpartitions *number*
Defines the maximum number of partitions the disklabels for the considered architecture can hold. This statement cannot be repeated and should only appear in the std.${ARCH} file.

maxusers *min default max*
Indicates the range of values that will later be accepted by config(1) for the **maxusers** statement in the options selection part of the configuration file. In case the user doesn't include a **maxusers** statement in the configuration file, the value *default* is used instead.

device *base* [{*locators*}] [: dependencies]
Declares a device of name *base*. The optional list of *locators*, which can also be empty, indicates the device can have children attached directly to it. Internally, that means *base* becomes an *interface attribute*. For every device the user selects, config(1) will add the matching **CFDRIVER_DECL**() statement to ioconf.c. However, it is the responsibility of the developer to add the relevant **CFATTACH_DECL**() line to the source of the device's driver.

attach *base* **at** *attr* [, *attr* [, . . .]] [**with** *name*] [: dependencies]
All devices must have at least one declared attachment. Otherwise, they will never be found in the autoconf(9) process. The attributes on which an instance of device *base* can attach must be *interface attributes*, or **root** in case the device is at the top-level, which is usually the case of e.g., mainbus(4). The instances of device *base* will later attach to one interface attribute from the specified list.

Different **attach** definitions must use different names using the **with** option. It is then possible to use the associated *name* as a conditional element in a **file** statement.

defpseudo *base* [: dependencies]
Declares a pseudo-device. Those devices don't need an attachment to be declared, they will always be attached if they were selected by the user.

defpseudodev *base* [{*locators*}] [: dependencies]
Declares a pseudo-device. Those devices don't need an attachment to be declared, they will always be attached if they were selected by the user. This declaration should be used if the pseudodevice uses autoconf(9) functions to manage its instances or attach children. As for normal devices, an optional list of *locators* can be defined, which implies an interface attribute named *base*, allowing the pseudo-device to have children. Interface attributes can also be defined in the *dependencies* list.

file *path* [*condition*] [**needs-count**] [**needs-flag**] [**compile with** *rule*]
Adds a source file to the list of files to be compiled into the kernel, if the *conditions* are met. The **needs-count** option indicates that the source file requires the number of all the countable objects it depends on (through the *conditions*) to be defined. It is usually used for *pseudo-devices* whose number can be specified by the user in the **pseudo-device** statement. Countable objects are devices and pseudo-devices. For the former, the count is the number of declared instances. For the latter, it is the number specified by the user, defaulting to 1. The **needs-flag** options requires that a flag indicating the selection of an attribute to be created, but the precise number isn't needed. This is useful for source files that only partly depend on the attribute, and thus need to add pre-processor statements for it.

needs-count and **needs-flag** both produce a header file for each of the considered attributes. The name of that file is `<attribute>.h`. It contains one pre-processor definition of `NATTRIBUTE` set to 0 if the attribute was not selected by the user, or to the number of instances of the device in the **needs-count** case, or to 1 in all the other cases.

The *rule* argument specifies the `make`(1) rule that will be used to compile the source file. If it is not given, the default rule for the type of the file will be used. For a given file, there can be more than one **file** statement, but not from the same configuration source file, and all later statements can only specify a *rule* argument, and no *conditions* or flags. This is useful when a file needs special consideration from one particular architecture.

object *path* [*condition*]
Adds an object file to the list of objects to be linked into the kernel, if the *conditions* are met. This is most useful for third parties providing binary-only components.

device-major *base* [**char** *number*] [**block** *number*] [*condition*]
Associates a major device number with the device *base*. A device can be a character device, a block device, or both, and can have different numbers for each. The *condition* indicates when the relevant line should be added to `ioconf.c`, and works just like the **file** statement.

makeoptions *condition name*+=*value* [, *condition name*+=*value*]
Appends to a definition in the generated `Makefile`.

OPTIONS SELECTION

machine *machine* [*arch* [*subarch* [`...`]]]
The **machine** statement should appear first in the kernel configuration file, with the exception of context-neutral statements. It makes `config`(1) include, in that order, the following files:

1. `conf/files`
2. `arch/${ARCH}/conf/files.${ARCH}` if defined
3. `arch/${SUBARCH}/conf/files.${SUBARCH}` for each defined sub-architecture
4. `arch/${MACHINE}/conf/files.${MACHINE}`

It also defines an attribute for the *machine*, the *arch* and each of the *subarch*.

package *path*
Simpler version of:

```
prefix PATH
include FILE
prefix
```

ident *string*
Defines the indentification string of the kernel. This statement is optional, and the name of the main configuration file will be used as a default value.

maxusers *number*
Despite its name, this statement does not limit the maximum number of users on the system. There is no such limit, actually. However, some kernel structures need to be adjusted to accommodate with more users, and the **maxusers** parameter is used for example to compute the maximum number of opened files, and the maximum number of processes, which itself is used to adjust a few other parameters.

options *name* [= *value*] [, *name* [= *value*], *...*]
Selects the option *name*, affecting it a *value* if the options requires it (see the **defflag** and **defparam** statements).

If the option has not been declared in the options description part of the kernel configuration machinery, it will be added as a pre-processor definition when source files are compiled.

no options *name* [, *name* [, *...*]]
Un-selects the option *name*. If option *name* has not been previously selected, the statement produces an error.

[**no**] **file-system** *name* [, *name* [, *...*]]
Adds or removes support for all the listed file-systems. A kernel must have support for at least one file-system.

config *name* **root on** *device* [**type** *fs*] [**dumps on** *device*]
Adds *name* to the list of kernel binaries to compile from the configuration file, using the specified root and dump devices information.

Any of the *device* and *fs* parameters can be wildcarded with "?" to let the kernel automatically discover those values.

At least one **config** statement must appear in the configuration file.

no config *name*
Removes *name* from the list of kernel binaries to compile from the configuration file.

instance **at** *attachment* [*locator specification*]
Configures an instance of a device attaching at a specific location in the device tree. All parameters can be wildcarded, with a "∗" for *instance*, and a "?" for *attachment* and the locators.

no *instance* [**at** *attachment*]
Removes the previously configured instances of a device that exactly match the given specification. If two instances differ only by their locators, both are removed. If no *attachment* is specified, all matching instances are removed.

If *instance* is a bare device name, all the previously defined instances of that device, regardless of the numbers or wildcard, are removed.

no device at *attachment*
Removes all previously configured instances that attach to the specified attachment. If *attachment* ends with a "∗", all instances attaching to all the variants of *attachment* are removed.

pseudo-device *device* [*number*]
Adds support for the specified pseudo-device. The parameter *number* is passed to the initialisation function of the pseudo-device, usually to indicate how many instances should be created. It defaults to 1, and some pseudo-devices ignore that parameter.

no pseudo-device *name*
Removes support for the specified pseudo-device.

makeoptions *name*=value [, *name*+=value [, . . .]]
Adds or appends to a definition in the generated `Makefile`. A definition cannot be overriden, it must be removed before it can be added again.

no makeoptions *name* [, *name* [, . . .]]
Removes one or more definitions from the generated `Makefile`.

FILES

The files are relative to the kernel source top directory (e.g., `/usr/src/sys`).

`arch/${MACHINE}/conf/std.${MACHINE}`	Standard configuration for the given architecture. This file should always be included.
`arch/${MACHINE}/conf/GENERIC`	Standard options selection file for the given architecture. Users should always start changing their main kernel configuration file by editing a copy of this file.
`conf/files`	Main options description file.

EXAMPLES

`config.samples`(5) uses several examples to cover all the practical aspects of writing or modifying a kernel configuration file.

SEE ALSO

`config`(1), `options`(4), `config.samples`(5), `config`(9)

NAME

config.samples — kernel configuration file syntax examples

DESCRIPTION

Devices, drivers and instances

For a given device, at most one driver will attach. In order for a driver to attach, the kernel configuration file must include a compatible instance of the driver for the location of the device. The following lines from the GENERIC kernel configuration file of NetBSD/i386 are examples of instances of drivers:

```
pchb*   at pci? dev ? function ?      # PCI-Host bridges
pcib*   at pci? dev ? function ?      # PCI-ISA bridges
ppb*    at pci? dev ? function ?      # PCI-PCI bridges

siop*   at pci? dev ? function ?      # Symbios 53c8xx SCSI
esiop*  at pci? dev ? function ?      # Symbios 53c875 SCSI and newer

ix0     at isa? port 0x300 irq 10     # EtherExpress/16
```

The first three instances allow three different drivers to attach to all the matching devices found on any PCI bus. This is the most generic case.

The next two lines allow two distinct drivers to attach to any matching device found on any PCI bus, but those two drivers are special because they both support some of the same devices. Each of the driver has a matching function that returns their score for the device that is being considered. autoconf(9) decides at run-time which driver will attach. Of course, it is deterministic so if the user wants to change the driver that attaches to the device, the instance of the other driver will have to be removed, e.g. by commenting it out.

The last line configures an instance of an ISA device. Unlike the PCI bus, the ISA bus cannot discover the devices that are present on the bus. The driver has to try accessing the device in order to discover it. That implies locators must be specified to some extent: a driver would usually need the base address of the device, some need the IRQ line that the device is configured to use, thoug some others would just try a set of known values, at the risk of badly interacting with other devices on the bus.

Hard-wiring kernel configuration

This technique consists in specifying exactly the location of the devices on a given system. In the general case it has very little use, because it does not change the size of the kernel, and it will prevent it from finding devices in case the hardware changes, even slightly.

Let's consider the following excerpt of dmesg(8) output:

```
auich0 at pci0 dev 31 function 5: i82801DB/DBM (ICH4/ICH4M) AC-97 Audio
```

The auich(4) driver (which controls Intel's AC-97 audio chips) attached there because of the following instance of GENERIC:

```
auich* at pci? dev ? function ?
```

Hard-wiring that instance means re-writing it to the following:

```
auich0 at pci0 dev 31 function 5
```

and that way, *auich0* will attach to that specific location, or will not attach.

Removing options and drivers

When two kernel configurations differ by a very small number of changes, it is easier to manage them by having one include the other, and add or remove the differences. Removing options and drivers is also useful in the situation of a user who wants to follow the development of NetBSD: drivers and options get added to

the configuration files found in the source tree, such as `GENERIC`, so one can include it and remove all options and drivers that are not relevant to the considered system. Additions to `GENERIC` will then automatically be followed and used in case they are relevant.

While negating an options (with **`no options`**) is unambiguous, it is not as clear for devices instances.

The **`no`** *`instance definition`* statements of `config`(1) syntax only apply on the current state of the configuration file, not on the resulting kernel binary. `autoconf`(9) has no knowledge of instance negation, thus it is currently impossible to express the following in a kernel configuration file:

> "I want support for `ath`(4) attaching at `pci`(4), but I do not want any instance of `ath`(4) attaching at *`pci3`*."

For a real-world use of **`no device at`** *`instance`* consider the following, taken from NetBSD/i386:

```
include "arch/i386/conf/GENERIC"

acpi0 at mainbus?

com* at acpi?
[... more instances of legacy devices attaching at acpi? ...]

no device at isa0
```

One could actually live without the *`isa0`* instance, as all the legacy devices are attached at *`acpi0`*. But unfortunately, dependencies on the *`isa`* attribute are not well registered all through the source tree, so an instance of the `isa`(4) driver is required to compile a kernel. So while:

```
no isa*
```

is what is intended, the `isa`(4) instance itself must be kept, and that is precisely the difference made by:

```
no device at isa0
```

Interface attributes

Interface attributes are a subtility of `config`(1) and `autoconf`(9), which often confuses users and utilities that parse `dmesg`(8) output to manipulate kernel configuration files. What they are is best shown by the following example.

The `dmesg`(8) output look like this:

```
auvia0 at pci0 dev 17 function 5: VIA Technologies VT8235 AC'97 Audio (rev 0x50)
audio0 at auvia0: full duplex, mmap, independent
```

while the kernel configuration look like this:

```
auvia* at pci? dev ? function ?
audio* at audiobus?
```

It is not obvious from the kernel configuration file that an `audio`(4) device can attach at an `auvia`(4) device. *`audiobus`* is an *interface attribute*, exposed by *`auvia`*.

Of course, it is possible to specify

```
audio* at auvia?
```

in the kernel configuration file, but then one instance per audio controler would be needed. *Interface attributes* reflect the fact there is a standard way to attach a device to its parent, no matter what the latter is precisely. It also means lower maintainance of the kernel configuration files because drivers for audio controlers are added more easily.

Most attachments are done through *interface attributes*, although only a few of them are specified that way in the configuration files found in the tree. Another example of such an attribute is `ata`:

```
viaide0 at pci0 dev 17 function 1
atabus0 at viaide0 channel 0

viaide* at pci? dev ? function ?
atabus* at ata?
```

SEE ALSO

`config`(1), `options`(4), `config`(5), `dmesg`(8)

NAME

core — memory image file format

SYNOPSIS

```
#include <sys/param.h>
```

For a.out-format core files:

```
#include <sys/core.h>
```

For ELF-format core files:

```
#include <sys/exec.h>
#include <sys/exec_elf.h>
```

DESCRIPTION

A small number of signals which cause abnormal termination of a process also cause a record of the process's in-core state to be written to disk for later examination by one of the available debuggers (see `signal`(7)).

This memory image is written to a file named from a per-process template; provided the terminated process had write permission, and provided the abnormality did not cause a system crash. (In this event, the decision to save the core file is arbitrary, see `savecore`(8).) The file is named from a per-process template, mapped to the sysctl variable *proc.<pid>.corename* (where <pid> has to be replaced by the pid in decimal format of the process). This template is either an absolute or relative path name, in which format characters can be used, preceded by the percent character (“%”). The following characters are recognized as format and substituted:

- **n** The process's name
- **p** The PID of the process (in decimal)
- **t** The process's creation date (a la `time`(3), in decimal)
- **u** The login name, as returned by `getlogin`(2)

By default, the per-process template string points to the default core name template, which is mapped to the sysctl variable *kern.defcorename*. Changing this value on a live system will change the core name template for all processes which didn't have a per-process template set. The default value of the default core name template is **%n.core** and can be changed at compile-time with the kernel configuration option **options DEFCORENAME** (see `options`(4))

The per-process template string is inherited on process creation, but is reset to point to the default core name template on execution of a set-id binary.

The maximum size of a core file is limited by `setrlimit`(2). Files which would be larger than the limit are not created.

ELF CORE FORMAT

ELF-format core files are described by a standard ELF exec header and a series of ELF program headers. Each program header describes a range of the virtual address space of the process.

In addition, NetBSD ELF core files include an ELF note section which provides additional information about the process. The first note in the note section has a note name of “NetBSD-CORE” and a note type of ELF_NOTE_NETBSD_CORE_PROCINFO (1), and contains the following structure:

```
struct netbsd_elfcore_procinfo {
   /* Version 1 fields start here. */
    uint32_t cpi_version;      /* netbsd_elfcore_procinfo version */
    uint32_t cpi_cpisize;      /* sizeof(netbsd_elfcore_procinfo) */
    uint32_t cpi_signo;        /* killing signal */
```

```
    uint32_t cpi_sigcode;      /* signal code */
    uint32_t cpi_sigpend[4];   /* pending signals */
    uint32_t cpi_sigmask[4];   /* blocked signals */
    uint32_t cpi_sigignore[4]; /* blocked signals */
    uint32_t cpi_sigcatch[4];  /* blocked signals */
    int32_t  cpi_pid;          /* process ID */
    int32_t  cpi_ppid;         /* parent process ID */
    int32_t  cpi_pgrp;         /* process group ID */
    int32_t  cpi_sid;          /* session ID */
    uint32_t cpi_ruid;         /* real user ID */
    uint32_t cpi_euid;         /* effective user ID */
    uint32_t cpi_svuid;        /* saved user ID */
    uint32_t cpi_rgid;         /* real group ID */
    uint32_t cpi_egid;         /* effective group ID */
    uint32_t cpi_svgid;        /* saved group ID */
    uint32_t cpi_nlwps;        /* number of LWPs */
    int8_t   cpi_name[32];     /* copy of p->p_comm */
    /* Add version 2 fields below here. */
};
```

The fields of *`struct netbsd_elfcore_procinfo`* are as follows:

cpi_version
: The version of this structure. The current version is defined by the NETBSD_ELFCORE_PROCINFO_VERSION constant.

cpi_cpisize
: The size of this structure.

cpi_signo
: Signal that caused the process to dump core.

cpi_sigcode
: Signal-specific code, if any, corresponding to *cpi_signo*.

cpi_sigpend
: A mask of signals pending delivery to the process. This may be examined by copying it to a *`sigset_t`*.

cpi_sigmask
: The set of signals currently blocked by the process. This may be examined by copying it to a *`sigset_t`*.

cpi_sigignore
: The set of signals currently being ignored by the process. This may be examined by copying it to a *`sigset_t`*.

cpi_sigcatch
: The set of signals with registers signals handlers for the process. This may be examined by copying it to a *`sigset_t`*.

cpi_pid
: Process ID of the process.

cpi_ppid
: Process ID of the parent process.

cpi_pgrp
: Process group ID of the process.

cpi_sid
: Session ID of the process.

cpi_ruid
: Real user ID of the process.

cpi_euid
: Effective user ID of the process.

cpi_svuid
: Saved user ID of the process.

cpi_rgid
: Real group ID of the process.

cpi_egid — Effective group ID of the process.

cpi_svgid — Saved group ID of the process.

cpi_nlwps — Number of kernel-visible execution contexts (LWPs) of the process.

cpi_name — Process name, copied from the p_comm field of *struct proc*.

The note section also contains additional notes for each kernel-visible execution context of the process (LWP). These notes have names of the form "NetBSD-CORE@nn" where "nn" is the LWP ID of the execution context, for example: "NetBSD-CORE@1". These notes contain register and other per-execution context data in the same format as is used by the ptrace(2) system call. The note types correspond to the ptrace(2) request numbers that return the same data. For example, a note with a note type of PT_GETREGS would contain a *struct reg* with the register contents of the execution context. For a complete list of available ptrace(2) request types for a given architecture, refer to that architecture's <machine/ptrace.h> header file.

A.OUT CORE FORMAT

The core file consists of a core header followed by a number of segments. Each segment is preceded by a core segment header. Both the core header and core segment header are defined in ⟨sys/core.h⟩.

The core header, *struct core*, specifies the lengths of the core header itself and each of the following core segment headers to allow for any machine dependent alignment requirements.

```
struct core {
    uint32_t c_midmag;         /* magic, id, flags */
    uint16_t c_hdrsize;        /* Size of this header (machdep algn) */
    uint16_t c_seghdrsize;     /* Size of a segment header */
    uint32_t c_nseg;           /* # of core segments */
    char      c_name[MAXCOMLEN+1];   /* Copy of p->p_comm */
    uint32_t c_signo;          /* Killing signal */
    u_long    c_ucode;          /* Signal code */
    u_long    c_cpusize;        /* Size of machine dependent segment */
    u_long    c_tsize;          /* Size of traditional text segment */
    u_long    c_dsize;          /* Size of traditional data segment */
    u_long    c_ssize;          /* Size of traditional stack segment */
};
```

The fields of *struct core* are as follows:

c_midmag — Core file machine ID, magic value, and flags. These values may be extracted with the **CORE_GETMID**(), **CORE_GETMAGIC**(), and **CORE_GETFLAG**() macros. The machine ID values are listed in ⟨sys/exec_aout.h⟩. For a valid core file, the magic value in the header must be COREMAGIC. No flags are defined for the core header.

c_hdrsize — Size of this data structure.

c_seghdrsize — Size of a segment header.

c_nseg — Number of segments that follow this header.

c_name — Process name, copied from the p_comm field of *struct proc*.

c_signo — Signal that caused the process to dump core.

c_ucode — Code associated with the signal.

c_cpusize — Size of the segment containing CPU-specific information. This segment will have the CORE_CPU flag set.

c_tsize — Size of the segment containing the program text.

c_dsize — Size of the segment containing the program's traditional data area.

c_ssize — Size of the segment containing the program's traditional stack area. This segment will have the CORE_STACK flag set.

The header is followed by *c_nseg* segments, each of which is preceded with a segment header, *struct coreseg*:

```
struct coreseg {
   uint32_t c_midmag;  /* magic, id, flags */
   u_long    c_addr;    /* Virtual address of segment */
   u_long    c_size;    /* Size of this segment */
};
```

The fields of *struct coreseg* are as follows:

c_midmag — Core segment magic value and flags. These values may be extracted with the **CORE_GETMAGIC**() and **CORE_GETFLAG**() macros. The magic value in the segment header must be CORESEGMAGIC. Exactly one of the flags CORE_CPU, CORE_DATA, or CORE_STACK will be set to indicate the segment type.

c_addr — Virtual address of the segment in the program image. Meaningless if the segment type is CORE_CPU.

c_size — Size of the segment, not including this header.

SEE ALSO

a.out(5), elf(5), gdb(1), setrlimit(2), sysctl(3), signal(7), sysctl(8)

HISTORY

A **core** file format appeared in Version 6 AT&T UNIX. The NetBSD a.out core file format was introduced in NetBSD 1.0. The NetBSD ELF core file format was introduced in NetBSD 1.6.

In releases previous to NetBSD 1.6, ELF program images produced a.out-format core files.

BUGS

There is no standard location or name for the CPU-dependent data structure stored in the CORE_CPU segment.

NAME

cpio — format of cpio archive files

DESCRIPTION

The **cpio** archive format collects any number of files, directories, and other file system objects (symbolic links, device nodes, etc.) into a single stream of bytes.

General Format

Each file system object in a **cpio** archive comprises a header record with basic numeric metadata followed by the full pathname of the entry and the file data. The header record stores a series of integer values that generally follow the fields in *struct stat*. (See `stat`(2) for details.) The variants differ primarily in how they store those integers (binary, octal, or hexadecimal). The header is followed by the pathname of the entry (the length of the pathname is stored in the header) and any file data. The end of the archive is indicated by a special record with the pathname "TRAILER!!!".

PWB format

XXX Any documentation of the original PWB/UNIX 1.0 format? XXX

Old Binary Format

The old binary **cpio** format stores numbers as 2-byte and 4-byte binary values. Each entry begins with a header in the following format:

```
struct header_old_cpio {
        unsigned short   c_magic;
        unsigned short   c_dev;
        unsigned short   c_ino;
        unsigned short   c_mode;
        unsigned short   c_uid;
        unsigned short   c_gid;
        unsigned short   c_nlink;
        unsigned short   c_rdev;
        unsigned short   c_mtime[2];
        unsigned short   c_namesize;
        unsigned short   c_filesize[2];
};
```

The *unsigned short* fields here are 16-bit integer values; the *unsigned int* fields are 32-bit integer values. The fields are as follows

magic The integer value octal 070707. This value can be used to determine whether this archive is written with little-endian or big-endian integers.

dev, *ino* The device and inode numbers from the disk. These are used by programs that read **cpio** archives to determine when two entries refer to the same file. Programs that synthesize **cpio** archives should be careful to set these to distinct values for each entry.

mode The mode specifies both the regular permissions and the file type. It consists of several bit fields as follows:

0170000 This masks the file type bits.

0140000 File type value for sockets.

0120000 File type value for symbolic links. For symbolic links, the link body is stored as file data.

0100000 File type value for regular files.
0060000 File type value for block special devices.
0040000 File type value for directories.
0020000 File type value for character special devices.
0010000 File type value for named pipes or FIFOs.
0004000 SUID bit.
0002000 SGID bit.
0001000 Sticky bit. On some systems, this modifies the behavior of executables and/or directories.
0000777 The lower 9 bits specify read/write/execute permissions for world, group, and user following standard POSIX conventions.

uid, gid The numeric user id and group id of the owner.

nlink The number of links to this file. Directories always have a value of at least two here. Note that hardlinked files include file data with every copy in the archive.

rdev For block special and character special entries, this field contains the associated device number. For all other entry types, it should be set to zero by writers and ignored by readers.

mtime Modification time of the file, indicated as the number of seconds since the start of the epoch, 00:00:00 UTC January 1, 1970. The four-byte integer is stored with the most-significant 16 bits first followed by the least-significant 16 bits. Each of the two 16 bit values are stored in machine-native byte order.

namesize
The number of bytes in the pathname that follows the header. This count includes the trailing NUL byte.

filesize The size of the file. Note that this archive format is limited to four gigabyte file sizes. See *mtime* above for a description of the storage of four-byte integers.

The pathname immediately follows the fixed header. If the **namesize** is odd, an additional NUL byte is added after the pathname. The file data is then appended, padded with NUL bytes to an even length.

Hardlinked files are not given special treatment; the full file contents are included with each copy of the file.

Portable ASCII Format

Version 2 of the Single UNIX Specification ("SUSv2") standardized an ASCII variant that is portable across all platforms. It is commonly known as the "old character" format or as the "odc" format. It stores the same numeric fields as the old binary format, but represents them as 6-character or 11-character octal values.

```
struct cpio_odc_header {
        char    c_magic[6];
        char    c_dev[6];
        char    c_ino[6];
        char    c_mode[6];
        char    c_uid[6];
        char    c_gid[6];
        char    c_nlink[6];
        char    c_rdev[6];
        char    c_mtime[11];
        char    c_namesize[6];
        char    c_filesize[11];
};
```

The fields are identical to those in the old binary format. The name and file body follow the fixed header. Unlike the old binary format, there is no additional padding after the pathname or file contents. If the files being archived are themselves entirely ASCII, then the resulting archive will be entirely ASCII, except for the NUL byte that terminates the name field.

New ASCII Format

The "new" ASCII format uses 8-byte hexadecimal fields for all numbers and separates device numbers into separate fields for major and minor numbers.

```
struct cpio_newc_header {
        char    c_magic[6];
        char    c_ino[8];
        char    c_mode[8];
        char    c_uid[8];
        char    c_gid[8];
        char    c_nlink[8];
        char    c_mtime[8];
        char    c_filesize[8];
        char    c_devmajor[8];
        char    c_devminor[8];
        char    c_rdevmajor[8];
        char    c_rdevminor[8];
        char    c_namesize[8];
        char    c_check[8];
};
```

Except as specified below, the fields here match those specified for the old binary format above.

magic The string “070701”.

check This field is always set to zero by writers and ignored by readers. See the next section for more details.

The pathname is followed by NUL bytes so that the total size of the fixed header plus pathname is a multiple of four. Likewise, the file data is padded to a multiple of four bytes. Note that this format supports only 4 gigabyte files (unlike the older ASCII format, which supports 8 gigabyte files).

In this format, hardlinked files are handled by setting the filesize to zero for each entry except the last one that appears in the archive.

New CRC Format

The CRC format is identical to the new ASCII format described in the previous section except that the magic field is set to “070702” and the *check* field is set to the sum of all bytes in the file data. This sum is computed treating all bytes as unsigned values and using unsigned arithmetic. Only the least-significant 32 bits of the sum are stored.

HP variants

The **cpio** implementation distributed with HPUX used XXXX but stored device numbers differently XXX.

Other Extensions and Variants

Sun Solaris uses additional file types to store extended file data, including ACLs and extended attributes, as special entries in cpio archives.

XXX Others? XXX

BUGS

The “CRC” format is mis-named, as it uses a simple checksum and not a cyclic redundancy check.

The old binary format is limited to 16 bits for user id, group id, device, and inode numbers. It is limited to 4 gigabyte file sizes.

The old ASCII format is limited to 18 bits for the user id, group id, device, and inode numbers. It is limited to 8 gigabyte file sizes.

The new ASCII format is limited to 4 gigabyte file sizes.

None of the cpio formats store user or group names, which are essential when moving files between systems with dissimilar user or group numbering.

Especially when writing older cpio variants, it may be necessary to map actual device/inode values to synthesized values that fit the available fields. With very large filesystems, this may be necessary even for the newer formats.

SEE ALSO

`cpio`(1), `tar`(5)

STANDARDS

The **`cpio`** utility is no longer a part of POSIX or the Single Unix Standard. It last appeared in Version 2 of the Single UNIX Specification (“SUSv2”). It has been supplanted in subsequent standards by `pax`(1). The portable ASCII format is currently part of the specification for the `pax`(1) utility.

HISTORY

The original cpio utility was written by Dick Haight while working in AT&T’s Unix Support Group. It appeared in 1977 as part of PWB/UNIX 1.0, the “Programmer’s Work Bench” derived from Version 6 AT&T UNIX that was used internally at AT&T. Both the old binary and old character formats were in use by 1980, according to the System III source released by SCO under their “Ancient Unix” license. The character format was adopted as part of IEEE Std 1003.1-1988 (“POSIX.1”). XXX when did "newc" appear? Who invented it? When did HP come out with their variant? When did Sun introduce ACLs and extended attributes? XXX

NAME

crontab − tables for driving cron

DESCRIPTION

A *crontab* file contains instructions to the *cron*(8) daemon of the general form: ‘‘run this command at this time on this date’’. Each user has their own crontab, and commands in any given crontab will be executed as the user who owns the crontab. Uucp and News will usually have their own crontabs, eliminating the need for explicitly running *su*(1) as part of a cron command.

Blank lines and leading spaces and tabs are ignored. Lines whose first non-space character is a pound-sign (#) are comments, and are ignored. Note that comments are not allowed on the same line as cron commands, since they will be taken to be part of the command. Similarly, comments are not allowed on the same line as environment variable settings.

An active line in a crontab will be either an environment setting or a cron command. An environment setting is of the form,

name = value

where the spaces around the equal-sign (=) are optional, and any subsequent non-leading spaces in *value* will be part of the value assigned to *name*. The *value* string may be placed in quotes (single or double, but matching) to preserve leading or trailing blanks. The *name* string may also be placed in quotes (single or double, but matching) to preserve leading, trailing or inner blanks.

Several environment variables are set up automatically by the *cron*(8) daemon. SHELL is set to /bin/sh, and LOGNAME and HOME are set from the /etc/passwd line of the crontab’s owner. HOME and SHELL may be overridden by settings in the crontab; LOGNAME may not.

(Another note: the LOGNAME variable is sometimes called USER on BSD systems... on these systems, USER will be set also.)

In addition to LOGNAME, HOME, and SHELL, *cron*(8) will look at MAILTO if it has any reason to send mail as a result of running commands in ‘‘this’’ crontab. If MAILTO is defined (and non-empty), mail is sent to the user so named. If MAILTO is defined but empty (MAILTO=""), no mail will be sent. Otherwise mail is sent to the owner of the crontab. This option is useful if you decide on /bin/mail instead of /usr/lib/sendmail as your mailer when you install cron -- /bin/mail doesn’t do aliasing, and UUCP usually doesn’t read its mail.

In order to provide finer control over when jobs execute, users can also set the environment variables CRON_TZ and CRON_WITHIN. The CRON_TZ variable can be set to an alternate time zone in order to affect when the job is run. Note that this only affects the scheduling of the job, not the time zone that the job perceives when it is run. If CRON_TZ is defined but empty (CRON_TZ=""), jobs are scheduled with respect to the local time zone.

The CRON_WITHIN variable should indicate the number of seconds within a job’s scheduled time that it should still be run. On a heavily loaded system, or on a system that has just been "woken up", jobs will sometimes start later than originally intended, and by skipping non-critical jobs because of delays, system load can be lightened. If CRON_WITHIN is defined but empty (CRON_WITHIN="") or set to some non-positive value (0, a negative number, or a non-numeric string), it is treated as if it was unset.

The format of a cron command is very much the V7 standard, with a number of upward-compatible extensions. Each line has five time and date fields, followed by a user name if this is the system crontab file, followed by a command. Commands are executed by *cron*(8) when the minute, hour, and month of year fields match the current time, *and* when at least one of the two day fields (day of month, or day of week) match the current time (see ‘‘Note’’ below). *cron*(8) examines cron entries once every minute. The time and date fields are:

field	allowed values
-----	----------
minute	0-59
hour	0-23
day of month	1-31

month	1-12 (or names, see below)
day of week	0-7 (0 or 7 is Sun, or use names)

A field may be an asterisk (*), which always stands for ‘‘first−last’’.

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an ‘‘hours’’ entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: ‘‘1,2,5,9’’, ‘‘0-4,8-12’’.

Step values can be used in conjunction with ranges. Following a range with ‘‘/<number>’’ specifies skips of the number’s value through the range. For example, ‘‘0-23/2’’ can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is ‘‘0,2,4,6,8,10,12,14,16,18,20,22’’). Steps are also permitted after an asterisk, so if you want to say ‘‘every two hours’’, just use ‘‘*/2’’.

Names can also be used for the ‘‘month’’ and ‘‘day of week’’ fields. Use the first three letters of the particular day or month (case doesn’t matter). Ranges or lists of names are not allowed.

The ‘‘sixth’’ field (the rest of the line) specifies the command to be run. The entire command portion of the line, up to a newline or % character, will be executed by /bin/sh or by the shell specified in the SHELL variable of the cronfile. Percent-signs (%) in the command, unless escaped with backslash (\), will be changed into newline characters, and all data after the first % will be sent to the command as standard input.

Note: The day of a command’s execution can be specified by two fields — day of month, and day of week. If both fields are restricted (ie, aren’t *), the command will be run when *either* field matches the current time. For example,
‘‘30 4 1,15 * 5’’ would cause a command to be run at 4:30 am on the 1st and 15th of each month, plus every Friday.

Instead of the first five fields, one of eight special strings may appear:

string	meaning
------	-------
@reboot	Run once, at startup.
@yearly	Run once a year, "0 0 1 1 *".
@annually	(same as @yearly)
@monthly	Run once a month, "0 0 1 * *".
@weekly	Run once a week, "0 0 * * 0".
@daily	Run once a day, "0 0 * * *".
@midnight	(same as @daily)
@hourly	Run once an hour, "0 * * * *".

EXAMPLE CRON FILE

```
# use /bin/sh to run commands, no matter what /etc/passwd says
SHELL=/bin/sh
# mail any output to ‘paul’, no matter whose crontab this is
MAILTO=paul
#
# run five minutes after midnight, every day
5 0 * * *       $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
# run at 2:15pm on the first of every month -- output mailed to paul
15 14 1 * *     $HOME/bin/monthly
# run at 10 pm on weekdays, annoy Joe
0 22 * * 1-5    mail -s "It’s 10pm" joe%Joe,%%Where are your kids?%
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am ..., everyday"
5 4 * * sun     echo "run at 5 after 4 every sunday"
```

SEE ALSO

cron(8), crontab(1)

EXTENSIONS

When specifying day of week, both day 0 and day 7 will be considered Sunday. BSD and ATT seem to disagree about this.

Lists and ranges are allowed to co-exist in the same field. "1-3,7-9" would be rejected by ATT or BSD cron -- they want to see "1-3" or "7,8,9" ONLY.

Ranges can include "steps", so "1-9/2" is the same as "1,3,5,7,9".

Names of months or days of the week can be specified by name.

Environment variables can be set in the crontab. In BSD or ATT, the environment handed to child processes is basically the one from /etc/rc.

Command output is mailed to the crontab owner (BSD can't do this), can be mailed to a person other than the crontab owner (SysV can't do this), or the feature can be turned off and no mail will be sent at all (SysV can't do this either).

All of the '@' commands that can appear in place of the first five fields are extensions.

AUTHOR

Paul Vixie <paul@vix.com>

NAME

cvs − Concurrent Versions System support files

NOTE

This documentation may no longer be up to date. Please consult the Cederqvist (CVS Manual) as specified in **cvs(1)**.

SYNOPSIS

$CVSROOT/CVSROOT/commitinfo,v

$CVSROOT/CVSROOT/cvsignore,v

$CVSROOT/CVSROOT/cvswrappers,v

$CVSROOT/CVSROOT/editinfo,v

$CVSROOT/CVSROOT/history

$CVSROOT/CVSROOT/loginfo,v

$CVSROOT/CVSROOT/modules,v

$CVSROOT/CVSROOT/rcsinfo,v

$CVSROOT/CVSROOT/taginfo,v

DESCRIPTION

cvs is a system for providing source control to hierarchical collections of source directories. Commands and procedures for using **cvs** are described in **cvs(1)**.

cvs manages *source repositories*, the directories containing master copies of the revision-controlled files, by copying particular revisions of the files to (and modifications back from) developers' private *working directories*. In terms of file structure, each individual source repository is an immediate subdirectory of **$CVSROOT**.

The files described here are supporting files; they do not have to exist for **cvs** to operate, but they allow you to make **cvs** operation more flexible.

You can use the ' modules ' file to define symbolic names for collections of source maintained with **cvs**. If there is no ' modules ' file, developers must specify complete path names (absolute, or relative to **$CVSROOT**) for the files they wish to manage with **cvs** commands.

You can use the ' commitinfo ' file to define programs to execute whenever ' **cvs commit** ' is about to execute. These programs are used for ''pre-commit'' checking to verify that the modified, added, and removed files are really ready to be committed. Some uses for this check might be to turn off a portion (or all) of the source repository from a particular person or group. Or, perhaps, to verify that the changed files conform to the site's standards for coding practice.

You can use the ' cvswrappers ' file to record **cvs** wrapper commands to be used when checking files into and out of the repository. Wrappers allow the file or directory to be processed on the way in and out of CVS. The intended uses are many, one possible use would be to reformat a C file before the file is checked in, so all of the code in the repository looks the same.

You can use the ' loginfo ' file to define programs to execute after any **commit**, which writes a log entry for changes in the repository. These logging programs might be used to append the log message to a file. Or send the log message through electronic mail to a group of developers. Or, perhaps, post the log message to a particular newsgroup.

You can use the ' taginfo ' file to define programs to execute after any **tag**or**rtag** operation. These programs might be used to append a message to a file listing the new tag name and the programmer who created it, or send mail to a group of developers, or, perhaps, post a message to a particular newsgroup.

You can use the ' rcsinfo ' file to define forms for log messages.

You can use the ' editinfo ' file to define a program to execute for editing/validating ' **cvs commit** ' log en-

tries. This is most useful when used with a ‘ rcsinfo ’ forms specification, as it can verify that the proper fields of the form have been filled in by the user committing the change.

You can use the ‘ cvsignore ’ file to specify the default list of files to ignore during **update**.

You can use the ‘ history ’ file to record the **cvs** commands that affect the repository. The creation of this file enables history logging.

FILES

modules

The ‘ modules ’ file records your definitions of names for collections of source code. **cvs** will use these definitions if you use **cvs** to check in a file with the right format to ‘ **$CVSROOT/CVSROOT/modules,v** ’.

The ‘ modules ’ file may contain blank lines and comments (lines beginning with ‘ **#** ’) as well as module definitions. Long lines can be continued on the next line by specifying a backslash (“\”) as the last character on the line.

A *module definition* is a single line of the ‘ modules ’ file, in either of two formats. In both cases, *mname* represents the symbolic module name, and the remainder of the line is its definition.

mname **−a** *aliases . . .*
This represents the simplest way of defining a module *mname*. The ‘ **−a** ’ flags the definition as a simple alias: **cvs** will treat any use of *mname* (as a command argument) as if the list of names *aliases* had been specified instead. *aliases* may contain either other module names or paths. When you use paths in *aliases*, ‘ **cvs checkout** ’ creates all intermediate directories in the working directory, just as if the path had been specified explicitly in the **cvs** arguments.

mname [*options*] *dir* [*files . . .*] [**&***module . . .*]

In the simplest case, this form of module definition reduces to ‘ *mname dir* ’. This defines all the files in directory *dir* as module *mname*. *dir* is a relative path (from **$CVSROOT**) to a directory of source in one of the source repositories. In this case, on **checkout**, a single directory called *mname* is created as a working directory; no intermediate directory levels are used by default, even if *dir* was a path involving several directory levels.

By explicitly specifying *files* in the module definition after *dir*, you can select particular files from directory *dir*. The sample definition for **modules** is an example of a module defined with a single file from a particular directory. Here is another example:

m4test unsupported/gnu/m4 foreach.m4 forloop.m4

With this definition, executing ‘ **cvs checkout m4test** ’ will create a single working directory ‘ m4test ’ containing the two files listed, which both come from a common directory several levels deep in the **cvs** source repository.

A module definition can refer to other modules by including ‘ **&***module* ’ in its definition. **checkout** creates a subdirectory for each such *module*, in your working directory.
New in **cvs** *1.3;* avoid this feature if sharing module definitions with older versions of **cvs**.

Finally, you can use one or more of the following *options* in module definitions:

‘ **−d** *name* ’, to name the working directory something other than the module name.
New in **cvs** *1.3;* avoid this feature if sharing module definitions with older versions of **cvs**.

‘ **−i** *prog* ’ allows you to specify a program *prog* to run whenever files in a module are committed. *prog* runs with a single argument, the full pathname of the affected directory in a source repository. The ‘ commitinfo ’, ‘ loginfo ’, and ‘ editinfo ’ files provide other ways to call a program on **commit**.

‘ **−o** *prog* ’ allows you to specify a program *prog* to run whenever files in a module are checked out. *prog* runs with a single argument, the module name.

‘ **−e** *prog* ’ allows you to specify a program *prog* to run whenever files in a module are exported. *prog* runs with a single argument, the module name.

‘ **−t** *prog* ’ allows you to specify a program *prog* to run whenever files in a module are tagged. *prog* runs with two arguments: the module name and the symbolic tag specified to **rtag**.

‘ **−u** *prog* ’ allows you to specify a program *prog* to run whenever ‘ **cvs update** ’ is executed from the top-level directory of the checked-out module. *prog* runs with a single argument, the full path to the source repository for this module.

commitinfo, **loginfo**, **rcsinfo**, **editinfo**

These files all specify programs to call at different points in the ‘ **cvs commit** ’ process. They have a common structure. Each line is a pair of fields: a regular expression, separated by whitespace from a filename or command-line template. Whenever one of the regular expression matches a directory name in the repository, the rest of the line is used. If the line begins with a **#** character, the entire line is considered a comment and is ignored. Whitespace between the fields is also ignored.

For ‘ loginfo ’, the rest of the line is a command-line template to execute. The templates can include not only a program name, but whatever list of arguments you wish. If you write ‘ **%s** ’ somewhere on the argument list, **cvs** supplies, at that point, the list of files affected by the **commit**. The first entry in the list is the relative path within the source repository where the change is being made. The remaining arguments list the files that are being modified, added, or removed by this **commit** invocation.

For ‘ taginfo ’, the rest of the line is a command-line template to execute. The arguments passed to the command are, in order, the *tagname* , *operation* (i.e. **add** for ‘tag’, **mov** for ‘tag -F’, and **del** for ‘tag -d‘), *repository* , and any remaining are pairs of **filename revision .** A non-zero exit of the filter program will cause the tag to be aborted.

For ‘ commitinfo ’, the rest of the line is a command-line template to execute. The template can include not only a program name, but whatever list of arguments you wish. The full path to the current source repository is appended to the template, followed by the file names of any files involved in the commit (added, removed, and modified files).

For ‘ rcsinfo ’, the rest of the line is the full path to a file that should be loaded into the log message template.

For ‘ editinfo ’, the rest of the line is a command-line template to execute. The template can include not only a program name, but whatever list of arguments you wish. The full path to the current log message template file is appended to the template.

You can use one of two special strings instead of a regular expression: ‘ **ALL** ’ specifies a command line template that must always be executed, and ‘ **DEFAULT** ’ specifies a command line template to use if no regular expression is a match.

The ‘ commitinfo ’ file contains commands to execute *before* any other **commit** activity, to allow you to check any conditions that must be satisfied before **commit** can proceed. The rest of the **commit** will execute only if all selected commands from this file exit with exit status **0**.

The ‘ rcsinfo ’ file allows you to specify *log templates* for the **commit** logging session; you can use this to provide a form to edit when filling out the **commit** log. The field after the regular expression, in this file, contains filenames (of files containing the logging forms) rather than command templates.

The ‘ editinfo ’ file allows you to execute a script *before the commit starts*, but after the log information is recorded. These "edit" scripts can verify information recorded in the log file. If the edit script exits with a non-zero exit status, the commit is aborted.

The ‘ loginfo ’ file contains commands to execute *at the end* of a commit. The text specified as a commit log message is piped through the command; typical uses include sending mail, filing an article in a newsgroup, or appending to a central file.

cvsignore, **.cvsignore**

The default list of files (or **sh**(**1**) file name patterns) to ignore during ‘ **cvs update** ’. At startup time, **cvs** loads the compiled in default list of file name patterns (see **cvs**(**1**)). Then the per-repository list included in **$CVSROOT/CVSROOT/cvsignore** is loaded, if it exists. Then the per-user list is loaded from ‘ $HOME/.cvsignore ’. Finally, as **cvs** traverses through your directories, it will load any per-directory ‘ .cvsignore ’ files whenever it finds one. These per-directory files are only valid for exactly the directory that contains them, not for any sub-directories.

history Create this file in **$CVSROOT/CVSROOT** to enable history logging (see the description of ‘ **cvs history** ’).

SEE ALSO

cvs(**1**),

COPYING

NAME

daily.conf — daily maintenance configuration file

DESCRIPTION

The **daily.conf** file specifies which of the standard /etc/daily services are performed. The /etc/daily script is run, by default, every night on a NetBSD system.

The variables described below can be set to "YES" or "NO" in the /etc/daily.conf file. Most default to "YES", but not all. Check the /etc/defaults/daily.conf file if you are in doubt. (Note that you should never edit /etc/defaults/daily.conf directly, as it is often replaced during system upgrades.)

find_core This runs find(1) over the entire local filesystem, looking for core files.

run_msgs This runs msgs(1) with the **-c** argument.

expire_news This runs the /etc/expire.news script.

purge_accounting This ages accounting files in /var/account.

run_calendar This runs calendar(1) with the **-a** argument.

check_disks This uses the df(1) and dump(8) to give disk status, and also reports failed raid(4) components.

show_remote_fs In check_disks, show remote file systems, which are not reported on by default.

check_mailq This runs mailq(1).

check_network This runs netstat(1) with the **-i** argument, and also checks the rwhod(8) database, and runs ruptime(1) if there are hosts in /var/rwho.

full_netstat By default, **check_network** outputs a summarized version of the netstat(1) report. If a full version of the output run with the **-inv** options is desired, set this variable.

run_fsck This runs fsck(8) with the **-n** option.

run_rdist This runs rdist(1) with /etc/Distfile.

run_security This runs the /etc/security script looking for possible security problems with the system.

run_skeyaudit Runs the skeyaudit(1) program to check the S/Key database and informs users of S/Keys that are about to expire.

The variables described below can be set to modify the tests:

find_core_ignore_fstypes
Lists filesystem types to ignore during the **find_core** phase. Prefixing the type with a '!' inverts the match. For example, procfs !local will ignore procfs type filesystems and filesystems that are not local.

send_empty_security
If set, the report generated by the **run_security** phase will always be sent, even if it is empty.

FILES

`/etc/daily`	daily maintenance script
`/etc/daily.conf`	daily maintenance configuration
`/etc/defaults/daily.conf`	default settings, overridden by `/etc/daily.conf`
`/etc/daily.local`	local site additions to `/etc/daily`

SEE ALSO

`monthly.conf`(5), `security.conf`(5), `weekly.conf`(5)

HISTORY

The **`daily.conf`** file appeared in NetBSD 1.3.

NAME

dhclient.conf - DHCP client configuration file

DESCRIPTION

The dhclient.conf file contains configuration information for *dhclient*, the Internet Systems Consortium DHCP Client.

The dhclient.conf file is a free-form ASCII text file. It is parsed by the recursive-descent parser built into dhclient. The file may contain extra tabs and newlines for formatting purposes. Keywords in the file are case-insensitive. Comments may be placed anywhere within the file (except within quotes). Comments begin with the # character and end at the end of the line.

The dhclient.conf file can be used to configure the behavior of the client in a wide variety of ways: protocol timing, information requested from the server, information required of the server, defaults to use if the server does not provide certain information, values with which to override information provided by the server, or values to prepend or append to information provided by the server. The configuration file can also be preinitialized with addresses to use on networks that don't have DHCP servers.

PROTOCOL TIMING

The timing behavior of the client need not be configured by the user. If no timing configuration is provided by the user, a fairly reasonable timing behavior will be used by default - one which results in fairly timely updates without placing an inordinate load on the server.

The following statements can be used to adjust the timing behavior of the DHCP client if required, however:

The **timeout** *statement*

timeout *time* **;**

The *timeout* statement determines the amount of time that must pass between the time that the client begins to try to determine its address and the time that it decides that it's not going to be able to contact a server. By default, this timeout is sixty seconds. After the timeout has passed, if there are any static leases defined in the configuration file, or any leases remaining in the lease database that have not yet expired, the client will loop through these leases attempting to validate them, and if it finds one that appears to be valid, it will use that lease's address. If there are no valid static leases or unexpired leases in the lease database, the client will restart the protocol after the defined retry interval.

The **retry** *statement*

retry *time***;**

The *retry* statement determines the time that must pass after the client has determined that there is no DHCP server present before it tries again to contact a DHCP server. By default, this is five minutes.

The **select-timeout** *statement*

select-timeout *time***;**

It is possible (some might say desirable) for there to be more than one DHCP server serving any given network. In this case, it is possible that a client may be sent more than one offer in response to its initial lease discovery message. It may be that one of these offers is preferable to the other (e.g., one offer may have the address the client previously used, and the other may not).

The *select-timeout* is the time after the client sends its first lease discovery request at which it stops waiting for offers from servers, assuming that it has received at least one such offer. If no offers have been received by the time the *select-timeout* has expired, the client will accept the first offer that arrives.

By default, the select-timeout is zero seconds - that is, the client will take the first offer it sees.

The **reboot** *statement*

reboot *time***;**

When the client is restarted, it first tries to reacquire the last address it had. This is called the INIT-REBOOT state. If it is still attached to the same network it was attached to when it last ran, this is the

quickest way to get started. The *reboot* statement sets the time that must elapse after the client first tries to reacquire its old address before it gives up and tries to discover a new address. By default, the reboot timeout is ten seconds.

The **backoff-cutoff** *statement*

backoff-cutoff *time***;**

The client uses an exponential backoff algorithm with some randomness, so that if many clients try to configure themselves at the same time, they will not make their requests in lock-step. The *backoff-cutoff* statement determines the maximum amount of time that the client is allowed to back off, the actual value will be evaluated randomly between 1/2 to 1 1/2 times the *time* specified. It defaults to two minutes.

The **initial-interval** *statement*

initial-interval *time***;**

The *initial-interval* statement sets the amount of time between the first attempt to reach a server and the second attempt to reach a server. Each time a message is sent, the interval between messages is incremented by twice the current interval multiplied by a random number between zero and one. If it is greater than the backoff-cutoff amount, it is set to that amount. It defaults to ten seconds.

LEASE REQUIREMENTS AND REQUESTS

The DHCP protocol allows the client to request that the server send it specific information, and not send it other information that it is not prepared to accept. The protocol also allows the client to reject offers from servers if they don't contain information the client needs, or if the information provided is not satisfactory.

There is a variety of data contained in offers that DHCP servers send to DHCP clients. The data that can be specifically requested is what are called *DHCP Options*. DHCP Options are defined in **dhcp-options(5)**.

The **request** *statement*

request [*option*] [**,** *... option*]**;**

The request statement causes the client to request that any server responding to the client send the client its values for the specified options. Only the option names should be specified in the request statement - not option parameters. By default, the DHCP server requests the subnet-mask, broadcast-address, time-offset, routers, domain-name, domain-name-servers and host-name options.

In some cases, it may be desirable to send no parameter request list at all. To do this, simply write the request statement but specify no parameters:

```
request;
```

The **require** *statement*

require [*option*] [**,** *... option]***;**

The require statement lists options that must be sent in order for an offer to be accepted. Offers that do not contain all the listed options will be ignored.

The **send** *statement*

send { [*option declaration*] [**,** *... option declaration*]**}**

The send statement causes the client to send the specified options to the server with the specified values. These are full option declarations as described in **dhcp-options(5)**. Options that are always sent in the DHCP protocol should not be specified here, except that the client can specify a **requested-lease-time** option other than the default requested lease time, which is two hours. The other obvious use for this statement is to send information to the server that will allow it to differentiate between this client and other clients or kinds of clients.

DYNAMIC DNS

The client now has some very limited support for doing DNS updates when a lease is acquired. This is prototypical, and probably doesn't do what you want. It also only works if you happen to have control

over your DNS server, which isn't very likely.

To make it work, you have to declare a key and zone as in the DHCP server (see **dhcpd.conf**(5) for details). You also need to configure the fqdn option on the client, as follows:

```
send fqdn.fqdn "grosse.fugue.com.";
send fqdn.encoded on;
send fqdn.server-update off;
```

The *fqdn.fqdn* option **MUST** be a fully-qualified domain name. You **MUST** define a zone statement for the zone to be updated. The *fqdn.encoded* option may need to be set to *on* or *off*, depending on the DHCP server you are using.

The **do-forward-updates** *statement*

do-forward-updates [*flag*] **;**

If you want to do DNS updates in the DHCP client script (see **dhclient-script(8)**) rather than having the DHCP client do the update directly (for example, if you want to use SIG(0) authentication, which is not supported directly by the DHCP client, you can instruct the client not to do the update using the **do-forward-updates** statement. *Flag* should be **true** if you want the DHCP client to do the update, and **false** if you don't want the DHCP client to do the update. By default, the DHCP client will do the DNS update.

The **omapi port** *statement*

omapi port [*port*] **;**

The **omapi port** statement causes the DHCP client to set up an OMAPI listener on the specified port. Only one such port should be specified. The OMAPI listener will be sensitive to connections from any IP address, so it is important to also set up an OMAPI key to protect the client from unauthorized changes.

The **omapi key** *statement*

omapi key [*key-id*] **;**

The **omapi key** statement causes the DHCP client to check any incoming OMAPI messages to make sure that they are signed by the specified key. If a message is not signed by this key, it is rejected with an "invalid key" error.

OPTION MODIFIERS

In some cases, a client may receive option data from the server which is not really appropriate for that client, or may not receive information that it needs, and for which a useful default value exists. It may also receive information which is useful, but which needs to be supplemented with local information. To handle these needs, several option modifiers are available.

The **default** *statement*

default [*option declaration*] **;**

If for some option the client should use the value supplied by the server, but needs to use some default value if no value was supplied by the server, these values can be defined in the **default** statement.

The **supersede** *statement*

supersede [*option declaration*] **;**

If for some option the client should always use a locally-configured value or values rather than whatever is supplied by the server, these values can be defined in the **supersede** statement.

The **prepend** *statement*

prepend [*option declaration*] **;**

If for some set of options the client should use a value you supply, and then use the values supplied by the server, if any, these values can be defined in the **prepend** statement. The **prepend** statement can only be used for options which allow more than one value to be given. This restriction is not enforced - if you ignore it, the behavior will be unpredictable.

The **append** *statement*

append [*option declaration*] **;**

If for some set of options the client should first use the values supplied by the server, if any, and then use values you supply, these values can be defined in the **append** statement. The **append** statement can only be used for options which allow more than one value to be given. This restriction is not enforced - if you ignore it, the behavior will be unpredictable.

LEASE DECLARATIONS

The **lease** *declaration*

lease { *lease-declaration* [*... lease-declaration]* **}**

The DHCP client may decide after some period of time (see **PROTOCOL TIMING**) that it is not going to succeed in contacting a server. At that time, it consults its own database of old leases and tests each one that has not yet timed out by pinging the listed router for that lease to see if that lease could work. It is possible to define one or more *fixed* leases in the client configuration file for networks where there is no DHCP or BOOTP service, so that the client can still automatically configure its address. This is done with the **lease** statement.

NOTE: the lease statement is also used in the dhclient.leases file in order to record leases that have been received from DHCP servers. Some of the syntax for leases as described below is only needed in the dhclient.leases file. Such syntax is documented here for completeness.

A lease statement consists of the lease keyword, followed by a left curly brace, followed by one or more lease declaration statements, followed by a right curly brace. The following lease declarations are possible:

bootp;

The **bootp** statement is used to indicate that the lease was acquired using the BOOTP protocol rather than the DHCP protocol. It is never necessary to specify this in the client configuration file. The client uses this syntax in its lease database file.

interface "*string*"**;**

The **interface** lease statement is used to indicate the interface on which the lease is valid. If set, this lease will only be tried on a particular interface. When the client receives a lease from a server, it always records the interface number on which it received that lease. If predefined leases are specified in the dhclient.conf file, the interface should also be specified, although this is not required.

fixed-address *ip-address***;**

The **fixed-address** statement is used to set the ip address of a particular lease. This is required for all lease statements. The IP address must be specified as a dotted quad (e.g., 12.34.56.78).

filename "*string*"**;**

The **filename** statement specifies the name of the boot filename to use. This is not used by the standard client configuration script, but is included for completeness.

server-name "*string*"**;**

The **server-name** statement specifies the name of the boot server name to use. This is also not used by the standard client configuration script.

option *option-declaration***;**

The **option** statement is used to specify the value of an option supplied by the server, or, in the case of predefined leases declared in dhclient.conf, the value that the user wishes the client configuration script to use if the predefined lease is used.

script "*script-name*"**;**

The **script** statement is used to specify the pathname of the dhcp client configuration script. This script is used by the dhcp client to set each interface's initial configuration prior to requesting an address, to test the

address once it has been offered, and to set the interface's final configuration once a lease has been acquired. If no lease is acquired, the script is used to test predefined leases, if any, and also called once if no valid lease can be identified. For more information, see **dhclient-script(8).**

vendor option space "*name***";**

The **vendor option space** statement is used to specify which option space should be used for decoding the vendor-encapsulate-options option if one is received. The *dhcp-vendor-identifier* can be used to request a specific class of vendor options from the server. See **dhcp-options(5)** for details.

medium "*media setup***";**

The **medium** statement can be used on systems where network interfaces cannot automatically determine the type of network to which they are connected. The media setup string is a system-dependent parameter which is passed to the dhcp client configuration script when initializing the interface. On Unix and Unix-like systems, the argument is passed on the ifconfig command line when configuring the interface.

The dhcp client automatically declares this parameter if it uses a media type (see the **media** statement) when configuring the interface in order to obtain a lease. This statement should be used in predefined leases only if the network interface requires media type configuration.

renew *date***;**

rebind *date***;**

expire *date***;**

The **renew** statement defines the time at which the dhcp client should begin trying to contact its server to renew a lease that it is using. The **rebind** statement defines the time at which the dhcp client should begin to try to contact *any* dhcp server in order to renew its lease. The **expire** statement defines the time at which the dhcp client must stop using a lease if it has not been able to contact a server in order to renew it.

These declarations are automatically set in leases acquired by the DHCP client, but must also be configured in predefined leases - a predefined lease whose expiry time has passed will not be used by the DHCP client.

Dates are specified as follows:

<*weekday*> <*year*>/<*month*>/<*day*> <*hour*>**:**<*minute*>**:**<*second*>

The weekday is present to make it easy for a human to tell when a lease expires - it's specified as a number from zero to six, with zero being Sunday. When declaring a predefined lease, it can always be specified as zero. The year is specified with the century, so it should generally be four digits except for really long leases. The month is specified as a number starting with 1 for January. The day of the month is likewise specified starting with 1. The hour is a number between 0 and 23, the minute a number between 0 and 59, and the second also a number between 0 and 59.

ALIAS DECLARATIONS

alias { *declarations ...* **}**

Some DHCP clients running TCP/IP roaming protocols may require that in addition to the lease they may acquire via DHCP, their interface also be configured with a predefined IP alias so that they can have a permanent IP address even while roaming. The Internet Systems Consortium DHCP client doesn't support roaming with fixed addresses directly, but in order to facilitate such experimentation, the dhcp client can be set up to configure an IP alias using the **alias** declaration.

The alias declaration resembles a lease declaration, except that options other than the subnet-mask option are ignored by the standard client configuration script, and expiry times are ignored. A typical alias declaration includes an interface declaration, a fixed-address declaration for the IP alias address, and a subnet-mask option declaration. A medium statement should never be included in an alias declaration.

OTHER DECLARATIONS

reject *ip-address***;**

The **reject** statement causes the DHCP client to reject offers from servers who use the specified address as a server identifier. This can be used to avoid being configured by rogue or misconfigured dhcp servers,

although it should be a last resort - better to track down the bad DHCP server and fix it.

interface "*name*" **{** *declarations ...* **}**

A client with more than one network interface may require different behavior depending on which interface is being configured. All timing parameters and declarations other than lease and alias declarations can be enclosed in an interface declaration, and those parameters will then be used only for the interface that matches the specified name. Interfaces for which there is no interface declaration will use the parameters declared outside of any interface declaration, or the default settings.

Note well: ISC dhclient only maintains one list of interfaces, which is either determined at startup from command line arguments, or otherwise is autodetected. If you supplied the list of interfaces on the command line, this configuration clause will add the named interface to the list in such a way that will cause it to be configured by DHCP. Which may not be the result you had intended. This is an undesirable side effect that will be addressed in a future release.

pseudo "*name*" "*real-name*" **{** *declarations ...* **}**

Under some circumstances it can be useful to declare a pseudo-interface and have the DHCP client acquire a configuration for that interface. Each interface that the DHCP client is supporting normally has a DHCP client state machine running on it to acquire and maintain its lease. A pseudo-interface is just another state machine running on the interface named *real-name*, with its own lease and its own state. If you use this feature, you must provide a client identifier for both the pseudo-interface and the actual interface, and the two identifiers must be different. You must also provide a separate client script for the pseudo-interface to do what you want with the IP address. For example:

```
interface "ep0" {
        send dhcp-client-identifier "my-client-ep0";
}
pseudo "secondary" "ep0" {
        send dhcp-client-identifier "my-client-ep0-secondary";
        script "/etc/dhclient-secondary";
}
```

The client script for the pseudo-interface should not configure the interface up or down - essentially, all it needs to handle are the states where a lease has been acquired or renewed, and the states where a lease has expired. See **dhclient-script(8)** for more information.

media "*media setup*" [**,** "*media setup*"**,** *...]***;**

The **media** statement defines one or more media configuration parameters which may be tried while attempting to acquire an IP address. The dhcp client will cycle through each media setup string on the list, configuring the interface using that setup and attempting to boot, and then trying the next one. This can be used for network interfaces which aren't capable of sensing the media type unaided - whichever media type succeeds in getting a request to the server and hearing the reply is probably right (no guarantees).

The media setup is only used for the initial phase of address acquisition (the DHCPDISCOVER and DHCPOFFER packets). Once an address has been acquired, the dhcp client will record it in its lease database and will record the media type used to acquire the address. Whenever the client tries to renew the lease, it will use that same media type. The lease must expire before the client will go back to cycling through media types.

SAMPLE

The following configuration file is used on a laptop running NetBSD 1.3. The laptop has an IP alias of 192.5.5.213, and has one interface, ep0 (a 3Com 3C589C). Booting intervals have been shortened somewhat from the default, because the client is known to spend most of its time on networks with little DHCP activity. The laptop does roam to multiple networks.

```
timeout 60;
retry 60;
```

```
reboot 10;
select-timeout 5;
initial-interval 2;
reject 192.33.137.209;

interface "ep0" {
   send host-name "andare.fugue.com";
   send dhcp-client-identifier 1:0:a0:24:ab:fb:9c;
   send dhcp-lease-time 3600;
   supersede domain-name "fugue.com rc.vix.com home.vix.com";
   prepend domain-name-servers 127.0.0.1;
   request subnet-mask, broadcast-address, time-offset, routers,
           domain-name, domain-name-servers, host-name;
   require subnet-mask, domain-name-servers;
   script "CLIENTBINDIR/dhclient-script";
   media "media 10baseT/UTP", "media 10base2/BNC";
}

alias {
  interface "ep0";
  fixed-address 192.5.5.213;
  option subnet-mask 255.255.255.255;
}
```

This is a very complicated dhclient.conf file - in general, yours should be much simpler. In many cases, it's sufficient to just create an empty dhclient.conf file - the defaults are usually fine.

SEE ALSO

dhcp-options(5), dhclient.leases(5), dhcpd(8), dhcpd.conf(5), RFC2132, RFC2131.

AUTHOR

dhclient(8) was written by Ted Lemon under a contract with Vixie Labs. Funding for this project was provided by Internet Systems Consortium. Information about Internet Systems Consortium can be found at **http://www.isc.org.**

Ì¾¾Î

dhclient.conf - DHCP ¥¯¥é¥¤¥¢¥ó¥ÈÀßÄê¥Õ¥¡¥¤¥ë

²òÀâ

dhclient.conf ¥Õ¥¡¥¤¥ë¤Ë¤Ï Internet Systems Consortium ¤Î DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ç¤¢¤ë *dhclient* ¤ÎÀßÄê¾ðÊó¤¬´Þ¤Þ¤ì¤Þ¤¹¡£

dhclient.conf ¤Ï¼«Í³·Á¼°¤Î ASCII ¥Æ¥¥¹¥È¥Õ¥¡¥¤¥ë¤Ç¤¹¡£ ¤³¤Î¥Õ¥¡¥¤¥ë¤Ï dhclient ¤ËÁÈ¤ß¹þ¤Þ¤ì¤¿ºÆµ¢²¼¹ß¥Ñ¡¼¥¶¤Ë²òÀÏ¤µ¤ì¤Þ¤¹¡£ ¥Õ¥¡¥¤¥ë¤Ë¤Ï¡¢À°·Á¤Î¤ÎÜÅª¤Ç¥¿¥Ö¤ä²þ¹Ô¤òÍ¾Ê¬¤Ë´Þ¤á¤ë¤³¤È¤â¤Ç¤¤Þ¤¹¡£ ¥Õ¥¡¥¤¥ëÃæ¤Î¥¡¼¥ï¡¼¥É¤Ç¤ÏÂçÊ¸»ú¾®Ê¸»ú¤ò¶èÊÌ¤·¤Þ¤»¤ó¡£ (¥¯¥©¡¼¥ÈÆâ¤Ï½ü¤¤¤Æ) ¥Õ¥¡¥¤¥ëÃæ¤Î¤É¤³¤Ç¤â¥·¥ã¡¼¥×¥µ¥¤¥ó # ¤Ç¥³¥á¥ó¥ÈÊ¸»ú¤Ç½ª¤ï¤ê¤Þ¤¹¡£

dhclient.conf ¥Õ¥¡¥¤¥ë¤Ç¡¢¥¯¥é¥¤¥¢¥ó¥È¤Î¤µ¤Þ¤¶¤Þ¤ÊÆ°ºî¤òÀßÄê¤Ç¤¤Þ¤¹¡£ ¤½¤ì¤é¤Ë¤Ï¡¢¥×¥í¥È¥³¥ë¤Î¥¿¥¤¥ß¥ó¥°¡¢¥µ¡¼¥Ð¤ËÂÐ¤·¤ÆÍ×µá¤¹¤ë¾ðÊó¡¢ ¥µ¡¼¥Ð¤ËÂÐ¤·¤ÆÉ¬¿Ü¤È¤µ¤ì¤ë¾ðÊó¡¢ ¥µ¡¼¥Ð¤¬¾ðÊó¤òÄó¶¡¤·¤Ê¤«¤Ã¤¿¾ì¹ç¤ËÍÑ¤¤¤ë¥Ç¥Õ¥©¥ë¥È¡¢ ¥µ¡¼¥Ð¤«¤éÄó¶¡¤µ¤ì¤¿¾ðÊó¤ò¾å½ñ¤¤¹¤ëÃÍ¡¢ ¥µ¡¼¥Ð¤«¤éÄó¶¡¤µ¤ì¤¿¾ðÊó¤ËÁ°ÃÖ¤ä¸åÃÖ¤¹¤ëÃÍ¤Ê¤É¤¬¤¢¤ê¤Þ¤¹¡£ ¤Þ¤¿¡¢DHCP ¥µ¡¼¥Ð¤ò»ý¤¿¤Ê¤¤¥Í¥Ã¥È¥ï¡¼¥¯¤Ç»È¤¦¥¢¥É¥ì¥¹¤Ç¤¢¤Ã¤Æ¤â¡¢ ¤¢¤é¤«¤¸¤áÀßÄê¥Õ¥¡¥¤¥ë¤Ç½é´ü²½¤¹¤ë¤³¤È¤â¤Ç¤¤Þ¤¹¡£

¥×¥í¥È¥³¥ë¤Î¥¿¥¤¥ß¥ó¥°

¥¯¥é¥¤¥¢¥ó¥È¤Î¥¿¥¤¥ß¥ó¥°Æ°ºî¤Ï¡¢¥æ¡¼¥¶¤¬¤Û¤È¤ó¤ÉÀßÄê¤¹¤ëÉ¬Í×¤Ï¤¢¤ê¤Þ¤»¤ó¡£ ¥æ¡¼¥¶¤¬¥¿¥¤¥ß¥ó¥°ÀßÄê¤ò¹Ô¤ï¤Ê¤±¤ì¤Ð¡¢ ¥µ¡¼¥Ð¤ËÌµÃá½ø¤ËÉé²Ù¤ò¤«¤±¤¿¤ê¤»¤º¤Ë¡¢»þ¹¿¤·¤¿¤¤¤Ê¤è¤¦¤Ê¡¢ ½¼Ê¬¤ËÅ¬ÀÚ¤Ê¥¿¥¤¥ß¥ó¥°Æ°ºî¤¬¥Ç¥Õ¥©¥ë¥È¤ÇÍÑ¤¤¤é¤ì¤Þ¤¹¡£

¤·¤«¤·¡¢É¬Í×¤Ë±þ¤¸¤Æ¡¢¼¡¤ÎÊ¸¤ò»ØÄê¤·¤Æ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Î¥¿¥¤¥ß¥ó¥°Æ°ºî¤òÄ´À°¤Ç¤¤Þ¤¹:

timeout *Ê¸*

timeout *time* **;**

timeout Ê¸¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬¥¢¥É¥ì¥¹¤ò·èÄê¤·»Ï¤á¤Æ¤«¤é¡¢ ¥µ¡¼¥Ð¤Ë¥¢¥¯¥»¥¹¤¹¤ë¤³¤È¤¬ ¤Ç¤¤Ê¤¤¤ÈÈ½ÃÇ¤¹¤ë¤Þ¤Ç¤Ë·Ð²á¤¹¤Ù¤»þ´Ö¤ò·èÄê¤·¤Þ¤¹¡£ ¥Ç¥Õ¥©¥ë¥È¤Ç¤Ï¤³¤Î¥¿¥¤¥à¥¢¥¦¥È¤ÏÃÍ¤Ï 60 ÉÃ¤Ç¤¹¡£ ¤³¤Î¥¿¥¤¥à¥¢¥¦¥È¤¬·²á¤®¤¿¸å¤Ï¡¢ ¤â¤·ÀÅÅª¤Ê¥ê¡¼¥¹¤¬ÀßÄê¥Õ¥¡¥¤¥ëÄêµÁ¤µ¤ì¤Æ¤¤¤ë¤«¡¢ ¥ê¡¼¥¹¥Ç¡¼¥¿¥Ù¡¼¥¹¤Ë¤Þ¤À´ü¸ÂÀÚ¤ì¤Ë¤Ê¤Ã¤Æ¤¤¤Ê¤¤¥ê¡¼¥¹¤¬»Ä¤Ã¤Æ¤¤¤ì¤Ð¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤Ï¤½¤ì¤é¤Î¥ê¡¼¥¹¤ò¤Ò¤È¤Ä¤º¤Ä¸¡¾Ú¤·¤Æ¤ß¤Æ¡¢ ¤¤¤º¤ì¤«¤¬Í¸ú¤Ê¤è¤¦¤Ç¤¢¤ì¤Ð¤½¤Î¥ê¡¼¥¹¤Î¥¢¥É¥ì¥¹¤ò»È¤¤¤Þ¤¹¡£ ¤â¤·ÀÅÅª¤Ê¥ê¡¼¥¹¤â¡¢¥ê¡¼¥¹¥Ç¡¼¥¿¥Ù¡¼¥¹Æâ¤Î´ü¸ÂÀÚ¤ì¤Æ¤¤¤Ê¤¤¥ê¡¼¥¹¤Ç Í¸ú¤Ê¤â¤Î¤â¸ºß¤·¤Ê¤±¤ì¤Ð¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤ÏÄêµÁ¤µ¤ì¤¿ retry ´Ö³Ö¤Î¸å¤Ç¥×¥í¥È¥³¥ë¤òºÆ³«¤·¤Þ¤¹¡£

retry *Ê¸*

retry *time***;**

retry Ê¸¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ DHCP ¥µ¡¼¥Ð¤¬Â¸ºß¤·¤Ê¤¤¤ÈÈ½ÃÇ¤·¤Æ¤«¤é ºÆ¤Ó DHCP ¥µ¡¼¥Ð¤Ë¥¢¥¯¥»¥¹¤ò»î¤ß¤ë¤Þ¤Ç¤Î´Ö¤Ë¡¢·Ð²á¤¹¤ë¤Ù¤»þ´Ö¤ò·èÄê¤·¤Þ¤¹¡£ ¥Ç¥Õ¥©¥ë¥È¤Ç¤Ï¡¢¤³¤ì¤Ï 5 Ê¬¤Ç¤¹¡£

select-timeout *Ê¸*

select-timeout *time***;**

¤¢¤ë¥Í¥Ã¥È¥ï¡¼¥¯¾å¤Ç¡¢Ê£¿ô¤Î DHCP ¥µ¡¼¥Ð¤¬¥µ¡¼¥Ó¥¹¤òÄó¶¡¤¹¤ë¤³¤È¤â¤Ç¤¤Þ¤¹ (¤½¤ÎÊý¤¬¤è¤¤¤È¸À¤¦¤Ò¤È¤â¤¤¤Þ¤¹)¡£ ¤½¤Î¾ì¹ç¡¢ºÇ½é¤Î¥ê¡¼¥¹È¯¹Ô¥á¥Ã¥»¡¼¥¸ (lease discovery message) ¤ØÎ±þÅú¤È¤·¤Æ¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬Ê£¿ô¤Î¥ê¡¼¥¹Äó¶¡¤ò¼õ¤±¤ë¤³¤È¤¬¤¢¤êÆÀ¤Þ¤¹¡£ ¤½¤ì¤é¤Î¦¤Á¡¢¤¢¤ëÄó¶¡¤¬Â¾¤ÎÄó¶¡¤è¤ê¤â¹¥¤Þ¤·¤¤¤«¤â¤·¤ì¤Þ¤»¤ó

(Îã¤¨¤Ð¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬°ÊÁ°»ÈÍÑ¤·¤Æ¤¤¤¿¥¢¥É¥ì¥¹¤¬¤¢¤ëÂó¶¡¤Ë´Þ¤Þ¤ì¤Æ¤¤¤ë¤¬¡¢ Â¾¤ÎÂó¶¡¤ËÏ´Þ¤Þ¤ì¤Ê¤¤¤Ê¤É)¡£

select-timeout ¤Ï¥¯¥é¥¤¥¢¥ó¥È¤¬ºÇ½é¤Î¥ê¡¼¥¹È¯¸«Í×µá ¤òÁ÷¿®¤·¤Æ¡¢ ¾¯¤Ê¤¯¤È¤â 1 ¤Ä¤ÎÂó¶¡¿½¤·½Ð¤ò¼õ¤±¤¿¾ì¹ç¡¢ ¥µ¡¼¥Ð¤«¤é¤ÎÂó¶¡¿½¤·½ÐÂÔ¤Á¤òä¤á¤ë¤Þ¤Ç¤Î»þ´Ö¤Ç¤¹¡£ ¤â¤· *select-timeout* ¤¬ÀÚ¤ì¤ë¤Þ¤Ç¤Ë¤É¤³¤«¤é¤âÂó¶¡¿½¤·½Ð¤ò¼õ¤±¼è¤ì¤Ê¤±¤ì¤Ð¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤Ï¤½¤Î¤¢¤ÈºÇ½é¤ËÅþÃå¤¹¤ëÂó¶¡¿½¤·½Ð¤ò¼õ¤±Æþ¤ì¤Þ¤¹¡£

¥Ç¥Õ¥©¥ë¥È¤Ç¤Ï¡¢select-timeout ÃÍ¤Ï 0 ÉÃ¤Ç¤¹¡£ ¤Ä¤Þ¤ê¥¯¥é¥¤¥¢¥ó¥È¤ÏºÇ½é¤Ë¼õ¤±¤è¤ëÂó¶¡¿½¤·½Ð¤ò¼õ¤±Æþ¤ì¤Þ¤¹¡£

reboot *Ê¸*

reboot *time***;**

¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢ºÆµ¯Æ°¤¹¤ë¤È¡¢ ºÇ¸å¤ËÊÝ»ý¤·¤Æ¤¤¤¿¥¢¥É¥ì¥¹¤ò¤Þ¤º¼èÆÀ¤·¤Ä¾¤½¤¦¤È¤·¤Þ¤¹¡£ ¤³¤ì¤ò INIT-REBOOT (½é´ü¥ê¥Ö¡¼¥È) ¾õÂÖ¤È¸Æ¤Ó¤Þ¤¹¡£ ºÇ¸å¤ËÆ°ºî¤·¤Æ¤¤¤¿¤È¤-¤ÈÆ±¤¸¥Í¥Ã¥È¥ï¡¼¥¯¤Ë ¥¯¥é¥¤¥¢¥ó¥È¤¬¤Þ¤ÀÀÜÂ³¤·¤Æ¤¤¤ì¤Ð¡¢¤³¤ì¤¬ºÇ¤âÁÇÁá¤¤µ¯Æ°ÊýË¡¤È¤Ê¤ê¤Þ¤¹¡£ *reboot* Ê¸¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ºÇ½é¤Ë¸Å¤¤¥¢¥É¥ì¥¹¤ÎºÆ¼èÆÀ¤ò»î¤ß¤Æ¤«¤é¡¢ ¤¢¤-¤é¤á¤Æ¿·¤·¤¤¥¢¥É¥ì¥¹¤òÈ¯¸«¤«¤·¤è¤¦¤È¤¹¤ë¤Þ¤Ç¤Ë¡¢ ·Ð²á¤¹¤Ù¤»þ´Ö¤òÀßÄê¤·¤Þ¤¹¡£ ¥Ç¥Õ¥©¥ë¥È¤Ç¤Ï¡¢reboot ¥¿¥¤¥à¥¢¥¦¥ÈÃÍ¤Ï 10 ÉÃ¤Ç¤¹¡£

backoff-cutoff *Ê¸*

backoff-cutoff *time***;**

¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢»Ø¿ôÅª¤Ê»þ´Ö»ÂàÈò (backoff) ¥¢¥ë¥´¥ê¥º¥à¤ò¡¢¤¢¤ëÄøÅÙ¤Î Íð¿ôÉÕ¤-¤Ç»ÈÍÑ¤·¤Þ¤¹¡£¤³¤ì¤Ï¡¢Â¿¤¯¤Î¥¯¥é¥¤¥¢¥ó¥È¤¬Æ±»þ¤Ë¼«Ê¬¤òÀßÄê¤·¤è¤¦¤È¤¹¤ë¾ì¹ç¤Ç¤â¡¢¥ê¥¯¥¨¥¹¥È¤¬¥í¥Ã¥¯¤·¤Æ¤·¤Þ¤¦¤³¤È¤¬¤Ê¤¤¤è¤¦¤Ë¤¹¤ë¤¿¤á¤Ç¤¹¡£ *backoff-cutoff* Ê¸¤Ï¡¢°ì»þÂàÈò¤Ëµö¤µ¤ì¤¿ºÇÂç»þ´Ö¤ò·èÄê¤·¤Þ¤¹¡£¥Ç¥Õ¥©¥ë¥ÈÃÍ¤Ï 2 Ê¬¤Ç¤¹¡£

initial-interval *Ê¸*

initial-interval *time***;**

initial-interval Ê¸¤Ï¡¢¥µ¡¼¥Ð¤Ø¤ÎºÇ½é¤Î¥¢¥¯¥»¥¹¤Î»î¤ß¤«¤é¼¡¤Î»î¤ß¤Þ¤Ç¤Î´Ö¤Î»þ´Ö¤ò ÀßÄê¤·¤Þ¤¹¡£¥á¥Ã¥»¡¼¥¸¤Î´Ö³Ö¤Ï¡¢¥á¥Ã¥»¡¼¥¸¤ò 1 ²óÁ÷¿®¤¹¤ë¤¿¤Ó¤Ë¡¢ ¸½ºß¤Î´Ö³Ö¤Ë 0 ¤«¤é 1 ¤Î´Ö¤ÎÍð¿ôÃÍ¤ò¾è¤¸¤¿¤â¤Î¤Î 2 ÇÜ¤ò¡¢¸½ºß¤Î´Ö³Ö¤Ë ²Ã¤¨¤¿¤â¤Î¤ËÊ¤ê¤Þ¤¹¡£ ¤³¤ÎÃÍ¤¬ backoff-cutoff ÃÍ¤èêÂç¤¤¯¤Ê¤ë¤È¡¢¤³¤Î»þ´Ö¤¬ÀßÄê¤µ¤ì¤Þ¤¹¡£ ¥Ç¥Õ¥©¥ë¥ÈÃÍ¤Ï 10 ÉÃ¤Ç¤¹¡£

¥ê¡¼¥¹Í×µá¤È¥ê¥¯¥¨¥¹¥È

DHCP ¥×¥í¥È¥³¥ë¤Ç¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤«¤é¥µ¡¼¥Ð¤ËÂÐ¤·¡¢ÆÃÄê¤Î¾ðÊó¤òÁ÷¤ë¤è¤¦ Í×µá¤·¤¿¤ê¡¢¼õ¤±Æþ¤½¤È÷¤Î¤Ç¤¤Æ¤¤¤Ê¤¤Â¾¤Î¾ðÊó¤ÏÁ÷¤é¤Ê¤¤¤è¤¦¤ËÍ×µá¤·¤¿¤ê ¤Ç¤¤Þ¤¹¡£ ¤Þ¿¡¢¥µ¡¼¥Ð¤«¤é¤ÎÂó¶¡¿½¤·½Ð¤Ë¥¯¥é¥¤¥¢¥ó¥È¤ÎÉ¬Í×¤Ê¾ðÊó¤¬´Þ¤Þ¤ì¤Ê¤¤ ¾ì¹ç¤ä¡¢Âó¶¡¤µ¤ì¤¿¾ðÊó¤¬½¼Ê¬¤Ç¤Ê¤¤¾ì¹ç¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬Âó¶¡¿½¤·½Ð¤ò µñÈÝ¤¹¤ë¤³¤È¤â¤Ç¤¤Þ¤¹¡£

DHCP ¥µ¡¼¥Ð¤¬ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤ËÁ÷¤ëÂó¶¡¿½¤·½Ð¤Ë´Þ¤Þ¤ì¤ë¥Ç¡¼¥¿¤Ë¤Ï¡¢ ¤µ¤Þ¤¶¤Þ¤Ê¤â¤Î¤¬¤¢¤ê¤Þ¤¹¡£ ÆÃ¤ËÍ×µá¤Ç¤¤ë¥Ç¡¼¥¿¤Ï *DHCP ¥ª¥×¥·¥ç¥ó* ¤È¸Æ¤Ð¤ì¤ë¤â¤Î¤Ç¤¹¡£ DHCP ¥ª¥×¥·¥ç¥ó¤Ï **dhcp-options(5)** ¤ËÄêµÁ¤µ¤ì¤Æ¤¤¤Þ¤¹¡£

request *Ê¸*

request [*option* **] [,** *... option* **];**

request Ê¸¤ò»ØÄê¤¹¤ë¤³¤È¤Ç¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢¥µ¡¼¥Ð¤ËÂÐ¤·¡¢¤½¤Î ¥¯¥é¥¤¥¢¥ó¥È¤Ë±þÅú¤¹¤ë¤Ê¤é¤Ð¡¢»ØÄê¤·¤¿¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤òÁ÷¤ë¤è¤¦ Í×µá¤¹¤ë¤è¤¦¤Ë¤Ê¤ê¤Þ¤¹¡£ request Ê¸¤Ë¤Ï¥ª¥×¥·¥ç¥ó¤Î¾¤À¤±¤ò»ØÄê¤·¡¢¥ª¥×¥·¥ç¥óÑ¥é¥á¡¼¥¿¤Ï»ØÄê¤·¤Þ¤»¤ó¡£ ¥Ç¥Õ¥©¥ë¥È¤Ç¤Ï DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï subnet-mask, broadcast-address, time-offset, routers, domain-name, domain-name-servers, host-name ¥ª¥×¥·¥ç¥ó¤òÍ×µá¤·¤Þ¤¹¡£

¾ì¹ç¤Ë¤è¤Ã¤Æ¤ÏÍ×µá¥ê¥¹¥È¤òÁ´¤¯Á÷¤é¤Ê¤¤¤³¤È¤¬¤³¤È¤¬Ë¾¤Þ¤·¤¤¤³¤È¤â¤¢¤¢¤ê¤Þ¤¹¡£ ¤½¤¦¤¹¤ë¤¿¤á¤Ë¤Ï¡¢Ã±½ã¤Ë¥Ñ¥é¥á¡¼¥¿¤ò»ØÄê¤·¤Ê¤¤ request Ê¸¤ò½ñ¤¤¤Æ²¼¤µ¤¤:

request;

require *Ê¸*

require [*option*] [**,** *... option]***;**

require Ê¸¤Ë¤Ï¡¢¤¢¤ëÄó¶¡¿½¤·½Ð¤ò¥¯¥é¥¤¥¢¥ó¥È¤¬¼õ¤±Æþ¤ì¤ë¤¿¤á¤Ë ¥µ¡¼¥Ð¤¬Á÷¤ë¤Ù¤-¥ª¥×¥·¥ç¥ó¤òÎóµó¤·¤Þ¤¹£ Îóµó¤µ¤ì¤¿¥ª¥×¥·¥ç¥ó¤¹¤Ù¤Æ¤ò´Þ¤Þ¤Ê¤¤Äó¶¡¿½¤·½Ð¤ÏÌµ»ë¤µ¤ì¤Þ¤¹£

send *Ê¸*

send { [*option declaration*] [**,** *... option declaration*]**}**

send Ê¸¤ò»ØÄê¤¹¤ë¤³È¤Ç¡¢¯¥é¥¤¥¢¥ó¥È¤Ï¡¢»ØÄê¤·¤¿¥ª¥×¥·¥ç¥ó¤ò»ØÄê¤·¤¿ÃÍ¤Ç¥µ¡¼¥Ð¤ËÁ÷¿®¤¹¤ë¤è¤¦¤Ë¤Ê¤ê¤Þ¤¹£ ¤³¤³¤Ç»ØÄê¤Ç¤-¤ë¥ª¥×¥·¥ç¥ó¤Ï¡¢ **dhcp-options(5)** ¤ÇÀâÌÀ¤µ¤ì¤Æ¤¤¤ë¥ª¥×¥·¥ç¥óÀë¸À¤¹¤Ù¤Æ¤Ç¤¹£ DHCP ¥×¥í¥È¥³¥ë¤Ç¾ï¤ËÁ÷¿é¤ì¤ë¥ª¥×¥·¥ç¥ó¤Ï ¤³¤³¤Ë»ØÄê¤¹¤ë¤Ù¤¤Ç¤Ï¤¢¤ê¤Þ¤»¤ó¡£Ã¢¤·¡¢ **requested-lease-time** ¥ª¥×¥·¥ç¥ó¤ò¥Ç¥Õ¥©¥ë¥È¤Î¥ê¡¼¥¹»þ´Ö (2 »þ´Ö) °Ê³°¤ÎÃÍ¤Ç»ØÄê¤¹¤ë¤³¤È¤Ï¤Ç¤-¤Þ¤¹£¤³¤ÎÊ¸¤ò»È¤¦¡Â¾Î¾ì¹ç¤È¤·¤ÆÌÀ¤é«¤Ê ¤â¤Î¤Ï¡¢¼«Ê¬¤È¤ÊÌ¼ïÎà¤Î¥¯¥é¥¤¥¢¥ó¥È¤È¤ò¶èÊÌ¤Ç¤-¤ë¤è¤¦¤Ê ¾ðÊó¤ò¡¢¥µ¡¼¥Ð¤ËÂÐ¤·Á÷¿®¤¹¤ë¾ì¹ç¤Ç¤¹£

Æ°Åª DNS

¸½ºß¡¢¥ê¡¼¥¹¤¬³ÍÆÀ¤µ¤ì¤¿ºÝ¤Ë DNS ¤Î¹¿·¤ò¹Ô¦¤¿¤á¤Î¡¢ È¾¾ïÈ¸ÂÄêÅª¤Ê¥µÝ¡¼¥È¤¬¥¯¥é¥¤¥¢¥ó¥È¤Ë¤¢¤ê¤Þ¤¹£ ¤³¤ì¤Ï¥×¥í¥È¥¿¥¤×Åª¤Ê¤â¤Î¤Ç¤¢¤ê¡¢ ¤ª¤½¤é¯¤¢¤Ê¤¿¤¬»×¤Ã¤Æ¤¤¤ë¤è¦¦¤Ë¤ÏÆ°¤¤¤Þ¤»¤ó¡£ ¤â¤·¡¢¤¢¤Ê¤¿¤¬¶öÁ³¤Ë¤â¼«Ê¬¤Î¤È¤³¤í¤Î DNS ¥µ¡¼¥Ð¤Î´ÉÍý¼Ô¤Ç¤¢¤ë¤È¤¤¦¦¤Ê¤é¡¢ ¤½¤Î¾ì¹ç¤Ë¸Â¤Ã¤Æ¤ÏÆ°¤-¤Þ¤¹£¤È¤Æ¤â¢¤ê¤½¦¤Ë¤Ê¤¤³¤È¤Ç¤¹¤¬¡£

¤³¤ì¤òÆ°ºî¤µ»¤ë¤¿¤á¤Ë¤Ï¡¢DHCP ¥µ¡¼¥Ð¤ÎÃæ¤Ç ¸°¤È¥¾¡¼¥ó¤òÀë¸À¤¹¤ëÉ¬Í×¤¬¤¢¤ê¤Þ¤¹ (¾ÜºÙ¤Ï **dhcpd.conf**(5) ¤ò»²¾È)¡£ ¤Þ¤¿¡¢¤Î¤è¦¤Ë¯¥é¥¤¥¢¥ó¥È¤Ç fqdn ¥ª¥×¥·¥ç¥ó¤òÀßÄê¤¹¤ëÉ¬Í×¤¬¤¢¤ê¤Þ¤¹:

```
send fqdn.fqdn "grosse.fugue.com.";
send fqdn.encoded on;
send fqdn.server-update off;
```

fqdn.fqdn ¥ª¥×¥·¥ç¥ó¤Ï **É¬¤º** ´°Á´¤Ê¥É¥á¥¤¥óÌ¾¤Ç¤Ê¤±¤ì¤Ð¤Ê¤ê¤Þ¤»¤ó¡£ ¹¿·¤¹¤ë¥¾¡¼¥óÉÂÐ¤¹¤ë¥¾¡¼¥óÊ¸¤ò **É¬¤º** Äê¤Á´¤Ê¤±¤ì¤Ð¤Ê¤ê¤Þ¤»¤ó¡£ *fqdn.encoded* ¥ª¥×¥·¥ç¥ó¤Ï¡¢»ÈÍÑ¤·¤Æ¤¤¤ë DHCP ¥µ¡¼¥Ð¤Ë¤è¤Ã¤Æ¤Ï¡¢ *on* ¤« *off* ¤ËÀßÄê¤¹¤ëÉ¬Í×¤¬¤¢¤ë¤«¤â¤·¤ì¤Þ¤»¤ó¡£

no-client-updates *Ê¸*

no-client-updates [*flag*] **;**

DHCP ¥¯¥é¥¤¥¢¥ó¥È¤¬Ä¾ÀÜ DNS ¤Î¹¿·¤ò¹Ô¦¤è¤ê¤â¡¢ DHCP ¥¯¥é¥¤¥¢¥ó¥È¥¯¥ê¥×¥È (**dhclient-script(8)** »²¾È) ¤ÎÃæ¤Ç DNS ¤Î¹¿·¤ò¹Ô¤¤¿¤¤¾ì¹ç (Îã¤¨¤Ð¡¢¤DHCP ¥¯¥é¥¤¥¢¥ó¥È¤¬Ä¾ÀÜ¥µ¥Ý¡¼¥È¤·¤Æ¤¤¤Ê¤¤ SIG(0) Ç§¾Ú¤ò»ÈÍÑ¤·¤¿¤¤¾ì¹ç) ¤Ë¤Ï¡¢ **no-client-updates** Ê¸¤ò»È¤Ã¤Æ¡¢¤¹¹¿·¤ò¹Ô¤Ê¤¤¤è¦¤Ë ¥¯¥é¥¤¥¢¥ó¥È¤Ë¶µ¨¤ë¤³¤È¤¬Ç¤Þ¤¹¡£ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤¬¹¿·¤¤¤ë¤³¤È¤òË¾¤Þ¤Ê¤¤¾ì¹ç¤Ï *flag* ¤ò **true** ¤Ë¤·¡¢ ¹¿·¤¤¤ë¤³¤È¤òË¾¤à¾ì¹ç¤Ï *flag* ¤ò **false** ¤Ë¤¹¤ë¤³¤È¤Ë¤Ê¤ê¤Þ¤¹¡£ ¥Ç¥Õ¥©¥ë¥È¤Ç¤Ï DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï DNS ¤Î¹¿·¤ò¹Ô¤¤¤Þ¤¹¡£

¥ª¥×¥·¥ç¥ó½¤¾þ»Ò

¤½¤Î¥¯¥é¥¤¥¢¥ó¥È¤Ë¤È¤Ã¤Æ¼Âº Ý¤Ë¤Ï¤Å¬ÀÚ¤Ç¤Ê¤¤
¥ª¥×¥·¥ç¥ó¥Ç¡¼¥¿¤ò¼õ¤±¼è¤Ã¤¿¤ê¡¢¤É¬Í×¤Ê¾ðÊó¤ò¼õ¤±¼è¤é¤Ê¤«¤Ã¤¿¤ê
¤¹¤ë¾ì¹ç¤Ç¡¢¤«¤Ä¡¢¤½¤ò¤é¤Î¾ðÊó¤ËÍøÍÑ²ÄÇ½¤Ê¥Ç¥Õ¥©¥ë¥È¤ÎÃÍ¤¬
¥¯¥é¥¤¥¢¥ó¥ÈÂ¦¤ËÂ¸º§¤¹¤ë¾ì¹ç¤¬¤¢¤ê¤Þ¤¹¡£
¤Þ¤¿¡¢ÍøÍÑ²ÄÇ½¤Ï¤«¥¥ë¥Î¾ðÊó¤ÇÊä¤¦É¬Í×¤Î¤¢¤ë¾ðÊó¤ò
¥¯¥é¥¤¥¢¥ó¥È¤¬¼õ¤±¤È¤ë¾ì¹ç¤â¢¤ê¤Þ¤¹¡£ ¤³¤¦¤¤¦¾ì¹ç¤ò°·¤¦¿¤á¤Ë¡¢¤¤¤¯¤Ä¤«¤ÎÆ°¤¤¤Ç¤¢¤ê¤Þ¤¹¡£

default *Ê¸*

default [*option declaration*] **;**

¤¢¤ë¥ª¥×¥·¥ç¥ó¤Ë¤Ä¤¤¤¤Æ¡¢ ¥µ¡¼¥Ð¤«¤éÄó¶¡¤µ¤ì¤ëÃÍ¤ò¥¯¥é¥¤¥¢¥ó¥È¤¬»È¤ï¤Ê¤±¤ì¤Ð¤Ê¤é¤Ê¤¤¾ì¹ç¡¢ ¤â¤·¥µ¡¼¥Ð¤«¤éÃÍ¤¬Äó¶¡¤µ¤ì¤Ê¤±¤ì¤Ð ²¿¤é¤«¤Î¥Ç¥Õ¥©¥ë¥ÈÃÍ¤ò»È¤¦É¬Í×¤¬¤¢¤ë¾ì¹ç¡¢ ¤½¤ì¤é¤ÎÃÍ¤ò **default** Ê¸¤ÇÄêµÁ¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£

supersede *Ê¸*

supersede [*option declaration* **] ;**

¤¢¤ë¥ª¥×¥·¥ç¥ó¤Ë¤Ä¤¤¤¤Æ¡¢ ¤É¤Î¤è¤¦¤ÊÃÍ¤¬¥µ¡¼¥Ð¤«¤éÄó¶¡¤µ¤ì¤Æ¤â¡¢ ¾ï¤Ë¥í¡¼¥«¥ë¤ÇÀßÄê¤µ¤ì¤¿ÃÍ¤ò»È¤ï¤Ê¤±¤ì¤Ð¤Ê¤é¤Ê¤¤¾ì¹ç¡¢ ¤½¤ì¤é¤ÎÃÍ¤ò **supersede** Ê¸¤ÇÄêµÁ¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£

prepend *Ê¸*

prepend [*option declaration* **] ;**

¤¢¤ë¥ª¥×¥·¥ç¥ó¤Î½¸¹ç¤Ë¤Ä¤¤¤¤Æ¡¢¤Þ¤º¥æ¡¼¥¶¤¬Äó¶¡¤¹¤ëÃÍ¤ò»È¤¤¡¢ ¤½¤Î¼¡¤Ë¥µ¡¼¥Ð¤«¤éÄó¶¡¤µ¤ì¤¿ÃÍ¤¬¤¢¤ì¤Ð¤½¤ì¤ò»È¤¦¾ì¹ç¡¢ ¤½¤ì¤é¤ÎÃÍ¤ò **prepend** Ê¸¤ÇÄêµÁ¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ **prepend** Ê¸¤ÏÊ£¿ô¤ÎÃÍ¤ò¼è¤ë¤³¤È¤Î¤Ç¤¤ë¥ª¥×¥·¥ç¥ó¤Ë¤ßÍÑ¤¤¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ ¤³¤Î©Ìó¤Ï¶¯À©¤µ¤ì¤ë¤â¤Î¤Ç¤Ï¤¢¤ê¤Þ¤»¤ó¤¬¡¢ ¤³¤ì¤òÌµ»ë¤·¤¿¾ì¹ç¡¢¤É¤Î¤è¤¦¤ÊµóÆ°¤Ë¤Ê¤ë¤«¤ÏÍ½ÁÛ¤Ç¤¤Þ¤»¤ó¡£

append *Ê¸*

append [*option declaration* **] ;**

¤¢¤ë¥ª¥×¥·¥ç¥ó¤Î½¸¹ç¤Ë¤Ä¤¤¤¤Æ¡¢¤Þ¤º¥µ¡¼¥Ð¤«¤éÄó¶¡¤µ¤ì¤¿ÃÍ¤ò»È¤¤¡¢ ¤½¤Î¼¡¤Ë¥æ¡¼¥¶¤¬Äó¶¡¤¹¤ëÃÍ¤¬¤¢¤ì¤Ð¤½¤ì¤â»È¤¦¾ì¹ç¡¢ ¤½¤ì¤é¤ÎÃÍ¤ò **append** Ê¸¤ÇÄêµÁ¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ **append** Ê¸¤ÏÊ£¿ô¤ÎÃÍ¤ò¼è¤ë¤³¤È¤Î¤Ç¤¤ë¥ª¥×¥·¥ç¥ó¤Ë¤ßÍÑ¤¤¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ ¤³¤Î©Ìó¤Ï¶¯À©¤µ¤ì¤ë¤â¤Î¤Ç¤Ï¤¢¤ê¤Þ¤»¤ó¤¬¡¢ ¤â¤·°ãÈ¿¤¹¤ë¤ÈÍ½´ü¤Ç¤¤Ê¤¤·ë²Ì¤È¤Ê¤ê¤Þ¤¹¡£

¥ê¡¼¥¹Àë¸À

lease *Àë¸À*

lease { *lease-declaration* [... *lease-declaration]* **}**

¤¢¤ë»þ´Ö **(¥×¥í¥È¥³¥ë¥¿¥¤¥ß¥ó¥°** »²¾È) ¤Î¸å¡¢DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï ¥µ¡¼¥Ð¤Ø¤Î¥¢¥¯¥»¥¹¤ËÀ®¸ù¤·¤½¤¦¤Ë¤Ê¤¤¤ÈÈ½ÃÇ¤¹¤ë¾ì¹ç¤¬¤¢¤ê¤Þ¤¹¡£ ¤½¤Î»þÅÀ¤Ç¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ï¼«Ê¬¤¬»ý¤Ã¤Æ¤¤¤ë¡¢¸Å¤¤¥ê¡¼¥¹¤Î¥Ç¡¼¥¿¥Ù¡¼¥¹¤ò ¸«¤Æ¡¢»þ´ÖÀÚ¤ì¤Ë¤Ê¤Ã¤Æ¤¤¤Ê¤¤¥ê¡¼¥¹¤ò½ç¤ËÄ´¤Ù¡¢¤½¤³¤Ëµó¤¬¤Ã¤Æ¤¤¤ë ¥ë¡¼¥¿¤Ë ping ¤ò¹Ô¤Ã¤Æ¡¢¤½¤ì¤¬ÍøÍÑ²ÄÇ½¤Ê¥ê¡¼¥¹¤«¤É¤¦¤«Ä´¤Ù¤Þ¤¹¡£ DHCP ¥µ¡¼¥Ó¥¹¤ä BOOTP ¥µ¡¼¥Ó¥¹¤¬Â¸ºß¤·¤Ê¤¤¥Í¥Ã¥È¥ï¡¼¥¯¤Î¤¿¤á¤Ë¡¢ 1 ¤Ä°Ê¾å¤Î *¸ÇÄê* ¥ê¡¼¥¹¤ò¥¯¥é¥¤¥¢¥ó¥ÈÀßÄê¥Õ¥¡¥¤¥ë¤ËÄêµÁ¤·¤Æ¤ª¤¤¤Æ¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬¥¢¥É¥ì¥¹¤ò¼«Æ°Åª¤ËÀßÄê¤Ç¤¤ë¤è¤¦¤Ë¤¹¤ë¤³¤È¤â¤Ç¤¤Þ¤¹¡£ ¤³¤ì¤Ï **lease** Ê¸¤Ç¹Ô¤¤¤Þ¤¹¡£

Ãí°Õ: lease Ê¸¤Ï¡¢DHCP ¥µ¡¼¥Ð¤«¤é¼õ¤±¼è¤Ã¤¿¥ê¡¼¥¹¤òµÏ¿¤¹¤ë¤¿¤á¤Ë¡¢ dhclient.leases ¥Õ¥¡¥¤¥ë¤Ç¤â»È¤ï¤ì¤Þ¤¹¡£ °Ê²¼¤ËÀâÌÀ¤¹¤ë¥ê¡¼¥¹ÍÑ¤Î¥·¥ó¥¿¥Ã¥¯¥¹¤Ë¤Ï dhclient.leases ¥Õ¥¡¥¤¥ë¤Ç¤Î¤ßÉ¬Í×¤Ê¤â¤Î¤â¤¢¤ê¤Þ¤¹¡£ ÀâÌÀ¤ò´°Á´¤Ê¤â¤Î¤Ë¤¹¤ë¤¿¤á¡¢¤½¤Î¤è¤¦¤Ê¥·¥ó¥¿¥Ã¥¯¥¹¤â¤³¤³¤ÇµÒ½Ò¤·¤Þ¤¹¡£

lease Ê¸¤Ï¡¢¥ê¡¼¥¹¥¡¼¥Ï¥É¡¢º¸Ãæ³ç¸Ì¡¢1 ¤Ä°Ê¾å¤Î¥ê¡¼¥¹Àë¸ÀÊ¸¡¢ ±¦Ãæ³ç¸Ì¤¬Â³¤¤¤¿¤â¤Î¤Ç¹½À®¤µ¤ì¤Þ¤¹¡£ ¥ê¡¼¥¹Àë¸À¤È¤·¤Æ¡¢¼¡¤Î¤â¤Î¤¬²ÄÇ½¤Ç¤¹:

bootp;

bootp Ê¸¤Ï¡¢¥ê¡¼¥¹¤¬ DHCP ¥×¥í¥È¥³¥ë¤Ç¤Ï¤Ê¤¯¡¢ BOOTP ¥×¥í¥È¥³¥ë¤òÍÑ¤¤¤ÆÄ¼èÆÀ¤µ¤ì¤¿¤³¤È¤ò¼¨¤·¤Þ¤¹¡£ ¤³¤ÎÊ¸¤ò¥¯¥é¥¤¥¢¥ó¥ÈÀßÄê¥Õ¥¡¥¤¥ë¤Ë»ØÄê¤¹¤ëÉ¬Í×¤ÏÁ´¤¯¤¢¤ê¤Þ¤»¤ó¡£ ¥¯¥é¥¤¥¢¥ó¥È¤Ï¤³¤Î¹½Ê¸¤ò¥ê¡¼¥¹¥Ç¡¼¥¿¥Ù¡¼¥¹¥Õ¥¡¥¤¥ëÆâ¤Ç»È¤¤¤Þ¤¹¡£

interface "*string***";**

interface ¥ê¡¼¥¹Ê¸¤Ï¡¢¤½¤Î¥ê¡¼¥¹¤òÍ¸ú¤È¤¹¤ë¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤ò¼¨¤·¤Þ¤¹¡£ ¤³¤ì¤¬ÀßÄê¤µ¤ì¤Æ¤¤¤ë¾ì¹ç¡¢¤³¤Î¥ê¡¼¥¹¤Ï¡¢»ØÄê¤µ¤ì¿¥¤¥ó¥¿¥Õ¥§¡¼¥¹ ¾å¤Ç¤Î¤ß»ÈÍÑ¤µ¤ì¤Þ¤¹¡£ ¥µ¡¼¥Ð¤«¤é¥ê¡¼¥¹¤ò¼õ¤±¼è¤Ã¤¿¤È¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤Ï¾ï¤Ë¤½¤Î¥ê¡¼¥¹¤ò¼õ¤±¼è¤Ã¤¿¥¤¥ó¥¿¥Õ¥§¡¼¥¹ÈÖ¹æ¤òµÏ¿¤·¤Þ¤¹¡£ dhclient.conf ¥Õ¥¡¥¤¥ë¤Ç»öÁ°¤Ë¥ê¡¼¥¹¤òÄêµÁ¤·¤Æ¤¤¤ë¾ì¹ç¡¢Í×µá¤µ¤ì¤Æ¤Ê¤¤ ¤Î¤Ç¤¹¤¬¡¢¤½¤Î¥ê¡¼¥¹¤Ç¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤â¢¤ïï¤Ð ¤Ê¤ê¤Þ¤»¤ó¡£

fixed-address *ip-address***;**

fixed-address Ê¸¤ÏÆÃÄê¤Î¥ê¡¼¥¹¤Î IP ¥¢¥É¥ì¥¹¤ò»ØÄê¤¹¤ëºÝ¤Ë»È¤¤¤Þ¤¹¡£ ¤³¤ì¤Ï¤¹¤Ù¤Æ¤Î lease Ê¸¤ËÉ¬Í×¤Ç¤¹¡£ IP ¥¢¥É¥ì¥¹¤Ï (12.34.56.78 ¤Î¤è¦Ë) ¥É¥Ã¥ÈÉÕ¤ 4 ¤ÄÁÈ·Á¼°¤Ç »ØÄê¤·¤Ê¤±¤ì¤ÐÐ¤Ê¤ê¤Þ¤»¤ó¡£

filename "*string*"**;**

filename Ê¸¤Ï»ÈÍÑ¤¹¤ë¥Ö¡¼¥È¥Õ¥¡¥¤¥ëÌ¾¤ò»ØÄê¤·¤Þ¤¹¡£ ¤³¤ì¤ÏÉ¸½àÅª¤Ê¥¯¥é¥¤¥¢¥óÈÀßÄê¥¹¥¯¥ê¥×¥È¤Ç¤Ï¥È¤ï¤ì¤Þ¤»¤ó¤¬¡¢ ÀâÌÀ¤Î´°Á´¤ò´ü¤¹¤¿¤á¤Ë¤³¤³¤Ë´Þ¤á¤Æ¤¢¤ê¤Þ¤¹¡£

server-name "*string*"**;**

server-name Ê¸¤Ï»ÈÍÑ¤¹¤ë¥Ö¡¼¥È¥µ¡¼¥ÐÌ¾¤ò»ØÄê¤·¤Þ¤¹¡£ ¤³¤ì¤âÉ¸½àÅª¤Ê¥¯¥é¥¤¥¢¥óÈÀßÄê¥¹¥¯¥ê¥×¥È¤Ç¤Ï¥È¤ï¤ì¤Þ¤»¤ó¡£

option *option-declaration***;**

option Ê¸¤Ï¡¢¥µ¡¼¥Ð¤«¤éÄó¶¡¤µ¤ì¤ë¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤ò»ØÄê¤¹¤ë¤Î¤Ë»È¤¤¤Þ¤¹¡£ ¤¢¤ë¤¤¤Ï¡¢dhclient.conf ¤Ç»öÁ°ÄêµÁ¥ê¡¼¥¹¤¬ÀëÀ¤µ¤ì¤Æ¤¤¤ë¾ì¹ç¤Ë¤Ï¡¢ ¤½¤Î»öÁ°ÄêµÁ¥ê¡¼¥¹¤¬»È¤ï¤ì¤ëºÝ¤Ë¥¯¥é¥¤¥¢¥óÈÀßÄê¥¹¥¯¥ê¥×¥È¤Ç»È¤ÍÑ¤·¤Æ Íß¤·¤¤ÃÍ¤ò»ØÄê¤·¤Þ¤¹¡£

script "*script-name*"**;**

script Ê¸¤Ï dhcp ¥¯¥é¥¤¥¢¥óÈÀßÄê¥¹¥¯¥ê¥×¥È¤Î¥Ñ¥¹Ì¾¤ò»ØÄê¤¹¤ë¤Î¤Ë»È¤¤¤Þ¤¹¡£ ¤³¤Î¥¹¥¯¥ê¥×¥È¤Ï¡¢¥¢¥É¥ì¥¹¤òÍ×µá¤·¤¿¤ê¡¢¥¢¥É¥ì¥¹¤Ë¤Ê¤Ã¤¿¤ê¤¹¤ë¤È¤¤Ë¼Â¹Ô¤µ¤ì¤ë¤è¤¦¤Ë¡¢ »î¤·¤¿¤ê¡¢ ¥ê¡¼¥¹¤òÌæÀ¤·¤Æ¤«¤é¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤Î½é´üÀßÄê¤ò¹Ô¤Ã¤¿¤ê¤¹¤ë¤Á°¤Ë¡¢ dhcp ¥¯¥é¥¤¥¢¥ó¥È¤¬³Æ¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤Î½é´üÀßÄê¤ò¹Ô¦¤Î¤Ë»È¤¤¤Þ¤¹¡£ ¥ê¡¼¥¹¤¬¼èÆÀ¤Ç¤-¤Ê¤«¤Ã¤¿¾ì¹ç¤Ë¤Ï¡¢ »öÁ°ÄêµÁ¥ê¡¼¥¹¤¬Â¸ºß¤¹¤ë¾ì¹ç¡¢¤½¤ì¤é¤ò»î¤¹¿¤á¤Ë¤³¤Î¥¯¥ê¥×¥È¤¬»È¤ï¤ì¤Þ¤¹¡£ ¤Þ¤¿¡¢¤Í-¸ú¤Ê¥ê¡¼¥¹¤¬¤Ò¤È¤Ä¤âÆÀ¤é¤ì¤Ê¤«¤Ã¤¿¾ì¹ç¤Ç¤â¡¢¤³¤Î¥¯¥ê¥×¥È¤Ï¡¢ 1 ²ó¤Ï¸Æ¤Ó½Ð¤µ¤ì¤Þ¤¹¡£ ¤è¤ê¾Ü¤·¤¯¡¢¤Ï¡¢**dhclient-script(8)** ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

vendor option space "*name*"**;**

vendor option space Ê¸¤Ï¡¢vendor-encapsulate-options ¥ª¥×¥·¥ç¥ó¤ò¼õ¿®¤·¤¿¾ì¹ç¡¢ Éü¹æ²½¤Ë¤É¤Î¥ª¥×¥·¥ç¥ó¶õ´Ö¤ò»ÈÍÑ¤¹¤ë¤Ù¤«¤ò»ØÄê¤¹¤ë¤¿¤á¤Ë»ÈÍÑ¤µ¤ì¤Þ¤¹¡£ ¥µ¡¼¥Ð¤«¤é¤Î¥Ù¥ó¥À¥ª¥×¥·¥ç¥ó¤ÎÆÃÄê¤Î¥¯¥é¥¤¥¢¥óÍ×µá¤¹¤ë¤¿¤á¤Ë¡¢ *dhcp-vendor-identifier* ¤ò»ÈÍÑ¤¹¤ë¤³¤È¤¬Ç¤¤Þ¤¹¡£ ¾Ü¤ºÙ¤Ï **dhcp-options(5)** ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

medium "*media setup*"**;**

medium Ê¸¤Ï¡¢ÀÜÂ³¤µ¤ì¤Æ¤¤¤ë¥Í¥Ã¥È¥ï¡¼¥¯¤Î¥¿¥×¤ò¥Í¥Ã¥È¥ï¡¼¥¯¤¬¼«Æ°Åª¤ËÈ½ÃÇ¤Ç¤¤Ê¤¤¤è¤¦¤Ê¥·¥¹¥Æ¥à¤Ç»È¤¦¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ Ê¸»úÍó media setup ¤Ï¥·¥¹¥Æ¥à°ÍÂ¸¤Î¥Ñ¥é¥á¡¼¥¿¡¢¥¢¥É¥ì¥¹¤Ç¡¢ dhcp ¥¯¥é¥¤¥¢¥óÈÀßÄê¥¹¥¯¥ê¥×¥È¤ËÅÏ¤µ¤ì¤Þ¤¹¡£ Unix ¤ª¤è¤Ó Unix É÷¤Î¥·¥¹¥Æ¥à¤Ç¤Ï¡¢ ¤³¤Î°ú¿ô¤Ï¥¤¥ó¥¿¥Õ¥§¡¼¥¹ÀßÄê¤¤ë¤È¤¤Ë ifconfig ¥³¥Þ¥ó¥É¥é¥¤¥ó¥Ë ÅÏ¤µ¤ì¤Þ¤¹¡£

¥ê¡¼¥¹¤òÆÀ¤ë¤¿¤á¤Ë¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤òÀßÄê¤¹¤ë ºÝ¤Ë¡¢dhcp ¥¯¥é¥¤¥¢¥óÈ¤¬¥á¥Ç¥£¥¢¡¿¥¤¥× (**media** Ê¸¤ò»²¾È) ¤ò»ÈÍÑ¤¹¤ë¾ì¹ç¡¢dhcp ¥¯¥é¥¤¥¢¥óÈ¤Ï¡¢¤³¤Î¥Ñ¥é¥á¡¼¥¿¤ò ¼«Æ°Åª¤ËÀë¸À¤·¤Þ¤¹¡£¥Í¥Ã¥È¥ï¡¼¥¯¤¬¥á¥Ç¥£¥¢¡¿¥¤¥×¤Î ÀßÄê¤òÉ¬Í×¤È¤¹¤ë¾ì¹ç¤Ï (¤¹¤ë¾ì¹ç¤Ë¸Â¤ê)¡¢¤³¤ÎÊ¸¤ò»öÁ°ÄêµÁ¥ê¡¼¥¹¤Ç »ÈÍÑ¤·¤Ê¤±¤ì¤Ð¤Ê¤ê¤Þ¤»¤ó¡£

renew *date***;**

rebind *date***;**

expire *date***;**

renew Ê¸¤Ï¡¢¸½º ß»ÈÍÑÃæ¤Î¥ê¡¼¥¹¤ò¹¹¿· (renew) ¤¹¤ë¤¿¤á¤Ë¡¢ dhcp ¥¯¥é¥¤¥¢¥ó¥È¤¬»ÈÍÑÃæ¤Î¥ê¡¼¥¹¤òÄó¶¡¤·¤Æ¤¯¤ì¤¿¥µ¡¼¥Ð¤Ø¤Î¥¢¥¯¥»¥¹¤Î »î¤ß¤ò³«»Ï¤·¤Ê¤±¤ì¤Ð¤Ê¤é¤Ê¤¤Æü»þ¤òÄêµÁ¤·¤Þ¤¹¡£**rebind** Ê¸¤Ï¡¢ ¥ê¡¼¥¹¤ò¹¹¿·¤¹¤ë¤¿¤á¤Ë¡¢dhcp ¥¯¥é¥¤¥¢¥ó¥È¤¬ ¤¤¤º¤ì¤«¤Î dhcp ¥µ¡¼¥Ð¤Ø¤Î¥¢¥¯¥»¥¹¤Î»î¤ß¤ò³«»Ï¤·¤Ê¤±¤ì¤Ð¤Ê¤é¤Ê¤¤Æü»þ¤òÄêµÁ¤·¤Þ¤¹¡£ **expire** Ê¸¤Ï¡¢¥ê¡¼¥¹¤Î¹¹¿·¤Î¤¿¤á¤Ë¥µ¡¼¥Ð¤Ë¥¢¥¯¥»¥¹¤Ç¤¤Ê¤«¤Ã¤¿¾ì¹ç¡¢ dhcp ¥¯¥é¥¤¥¢¥ó¥È¤¬¤½¤Î¥ê¡¼¥¹¤Î»ÈÍÑ¤òÄä»ß¤·¤Ê¤±¤ì¤Ð¤Ê¤é¤Ê¤¤Æü»þ¤ò ÄêµÁ¤·¤Þ¤¹¡£

¤³¤ì¤é¤ÎÀë¸À¤Ï¡¢DHCP ¥¯¥é¥¤¥¢¥ó¥È¤¬ÆÀ¤¿¥ê¡¼¥¹¤Æ¤Ç¤Ï¼«Æ°Å²¤ËÀßÄê¤µ¤ì¤Þ¤¹¡£ »öÁ°ÄêµÁ¥ê¡¼¥¹¤Î¦¤ÁDHCP ¥¯¥é¥¤¥¢¥ó¥È¤ËÍ¸ú¸Â¤¬²á®¤¿¤â¤Î¤ò»ÈÍÑ¤·¤Æ Íß¤·¤¯¤Ê¤¤¤â¤Î¤ÎÃæ¤Ç¤Ï¡¢¤³¤ì¤é¤ÎÀë¸À¤òÀßÄê¤·¤Æ¤ª¤¯É¬Í×¤¬¤¢¤ê¤Þ¤¹¡£

date ¤Ï°Ê²¼¤Î¤è¦¤Ë»ØÄê¤·¤Þ¤¹¡£

<weekday> *<year>*/*<month>*/*<day>* *<hour>*:*<minute>*:*<second>*

weekday ¤Ï¡¿Í´Ö¤¬,«¤Æ¥ê¡¼¥´ü¸Â¤ò¤ï¤«¤ê¤ä¤¹¯¤¹¤ë¤¿¤á¤ËÂ¸º§¤·¤Þ¤¹¡£ ¤³¤ì¤Ï¡¢0 ¤«¤é 6 ¤Þ¤Ç¤Î¿ô»ú¤Ç»ØÄê¤·¤Þ¤¹¡£0 ¤ÏÆüÍËÆü¤Ç¤¹¡£year ¤ÏÀ¤µª ¹þ¤ß¤Ç»ØÄê¤·¤Þ¤¹¡£¤Ç¤¹¤«¤é¡¢ËÜÅö¤ËÄ¹¤¤¥ê¡¼¥¹¤òÊÌ¤Ë¤¹¤ë¤È¡¢É¬¤º 4 ·å¤Ë ¤Ê¤ë¤Ï¤º¤Ç¤¹¡£month ¤Ï 1 (1 ·î¤òÉ½¤·¤Þ¤¹) ¤«¤é»Ï¤Þ¤ë¿ô»ú¤Ç»ØÄê¤·¤Þ¤¹¡£ day ¤ÏÆ±ÍÍ¤Ë 1 ¤«¤é»Ï¤Þ¤ë (·î¤Ë¤ª¤±¤ë) Æü¤È¤·¤Æ»ØÄê¤·¤Þ¤¹¡£hour ¤Ï¡¢ 0 ¤«¤é 23 ¤Î´Ö¤Î¿ô»ú¤Ç¤¹¡£minute ¤È second ¤Ï¤È¤â¤Ë 0 ¤«¤é 59 ¤Î´Ö¤Î ¿ô»ú¤ò»ØÄê¤·¤Þ¤¹¡£

¥¨¥¤¥ê¥¢¥¹À¸À

alias { *declarations ...* **}**

DHCP ¥¯¥é¥¤¥¢¥ó¥È¤¬ TCP/IP ¥í¡¼¥ß¥ó¥° (roaming) ¥×¥í¥È¥³¥ë¤ò¼Â¹Ô¤·¤Æ ¤¤¤ë¾ì¹ç¡¢DHCP ¤òÍÑ¤¤¤ÆÆÀ¤é¤ì¤ë¥ê¡¼¥¹¤À¤±Ç¤Ê¤¯¡¢»öÁ°¤ËÄêµÁ¤µ¤ì¤¿ IP ¥¨¥¤¥ê¥¢¥¹¤â¡¢¼«Ê¬¤¬»ÈÍÑ¤¹¤ë¥¿¥Õ¥§¡¼¥¹¤ËÀßÄê¤¹¤ëÉ¬Í×¤¬¤¢¤ë ¾ì¹ç¤¬¤¢¤ê¤Þ¤¹¡£Internet Systems Consortium ÈÇ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢ ¸ÇÄê¥¢¥É¥ì¥Ä¾ÀÜ»ØÄê¤Î¥í¡¼¥ß¥ó¥°¤ò¥µ¥Ý¡¼¥È·¤Æ¤¤Þ¤»¤ó¤¬¡¢¤½¤Î¼¡ï¤Î¾Â¸³ ¤¬¤Ç¤¤ë¤è¦¤Ë¡¢¤³¤Î dhcp ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢ **alias** Àë¸À¤ò»È¤Ã¤Æ IP ¥¨¥¤¥ê¥¢¥¹¤òÀßÄê¤¹¤ë¼èÈ÷¤Ï¤Ç¤¤Æ¤¤¤Þ¤¹¡£

alias Àë¸À¤Ï lease Àë¸À¤Ë»÷¤Æ¤¤¤Þ¤¹¡£Ã¢¤·¡¢É¸½à¤Î ¥¯¥é¥¤¥¢¥ó¥ÈÀßÄê¥¹¥¯¥ê¥×¥È¤Ç¤Ï¡¢subnet-mask ¥ª¥×¥·¥ç¥ó°Ê³°¤Î ¥ª¥×¥·¥ç¥ó¤È¡¢¾ï¤Í¸ú¸ÂÍ¸ú¸ü¸Â (expiry times) ¤ÌÌ»ë¤µ¤ì¤ëÅÀ¤¬Û¤Ê¤ê¤Þ¤¹¡£ ÉáÄÌ¤Î alias Àë¸À¤Ç¤Ï¡¢ interface Àë¸À¡¢IP ¥¨¥¤¥ê¥¢¥¹¤Î¤¿¤á¤Î ¸ÇÄê¥¢¥É¥ì¥Àë¸À¡¢subnet-mask ¥ª¥×¥·¥ç¥ó¤´Þ¤ß¤Þ¤¹¡£alias Àë¸À¤Ë¤Ï medium Ê¸¤Ï·è¤·¤Æ´Þ¤Þ¤ì¤Æ¤Ï¤Ê¤ê¤Þ¤»¤ó¡£

¤½¤ÎÂ¾¤ÎÀë¸À

reject *ip-address***;**

reject Ê¸¤Ë¤è¤ê¡¢DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï»ØÄê¤·¤¿¥¢¥É¥ì¥¹¤òµ¡¼¥Ð¼±ÊÌ»Ò¤È¤·¤Æ»ÈÍÑ¤¹¤ë ¥µ¡¼¥Ð¤«¤é¤ÎÄó¶¡¿½¤·½Ð¤òµñÈÝ¤¹¤ë¤è¦¤Ë¤Ê¤ê¤Þ¤¹¡£É¸½à¤Ë½àµò¤·¤Ê¤¤ dhcp ¥µ¡¼¥Ð¤äÀßÄê¤ò´Ö¤ã¤¨¤Æ¤¤¤ë dhcp ¥µ¡¼¥Ð¤Ë¤è¤Ã¤Æ¥¯¥é¥¤¥¢¥ó¥È¤¬ÀßÄê¤µ¤ì¤Ê¤¤ ¤è¦¤Ë¤¹¤ë¤¿¤á¤Ë¡¢¤³¤ÎÊ¸½ò¤ÈÍÑ¤¹¤ë¤³¤È¤¬Ç¤¤Þ¤¹¡£¤·¤«¤·Ê¬»¤¤ºÇ¸¸å¤ÎÉð´ï¤È¤¹¤ë¤Ù¤-¤Ç¤¹¡£¤³¤ì¤ËÀèÎ©¤Á¡¢Éå¤Ã¤¿ DHCP ¥µ¡¼¥Ð¤òÄÉ¤¤«¤±Æ ¤½¤ì¤ò¾¤¹Êý¤¬¤è¤¤¤Ç¤¹¡£

interface "*name*" **{** *declarations ...* **}**

Ê£¿ô¤Î¥Í¥Ã¥È¥ï¡¼¥¯¤¤¥ó¿¥Õ¥§¡¼¥¹¤ò»ý¤Ä¥¯¥é¥¤¥¢¥ó¥È¤Î¾ì¹ç¡¢DHCP ¤Ç ÀßÄê¤µ¤ì¤ë¥¤¥ó¿¥Õ¥§¡¼¥¹¤Ë¤è¤Ã¤Æ°Û¤Ê¤ëÆ°ºî¤ò¤µ¤»¤ëÉ¬Í×¤¬¤¢¤ë¾ì¹ç¬ ¤¢¤ê¤Þ¤¹¡£lease Àë¸À¤È alias Àë¸À¤ò½ü¤¯¤¹Ù¤Æ¤Î¥¿¥¤¥ß¥ó¥°¥Ñ¥é¥á¡¼¥¿ ¤ÈÀë¸À¤ò¡¢interface Àë¸À¤Ç°Ï¤à¤³¤È¬Ç¤¤Þ¤¹¡£¤½¤Î¾ì¹ç¡¢°Ï¤à¤³¤È¤¬¤Ç¤¤Þ¤»¤ó¤¬¡¢¤½¤Î ¥Ñ¥é¥á¡¼¥¿¤Ï»ØÄê¤·¤¿¿Ì¾°¤Ë¹ÇÃ×¤¹¤ë¥¤¥ó¿¥Õ¥§¡¼¥¹¤Ë¤Î¤ßÅ¬ÍÑ¤µ¤ì¤Þ¤¹¡£ interface Àë¸À¤ò»ý¤¿¸Ê¤¤¥¤¥ó¿¥Õ¥§¡¼¥¹¤Ï¡¢¤¹¤Ù¤Æ interface Àë¸À¤Î ³°Â¦¤ÇÀë¸À¤µ¤ì¤¿¥Ñ¥é¥á¡¼¥¿¡¢¤â¤¤¤·¤¯¯Ï¥Ç¥Õ¥©¥ë¥È¤ÎÀßÄê¤¬Å¬ÍÑ¤µ¤ì¤Þ¤¹¡£

pseudo "*name*" "*real-name*" **{** *declarations ...* **}**

¾õ¶·¤Ë¤è¤Ã¤Æ¤Ï²¾ÁÛ¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤òÀë¸À¤·¡¢ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤¬¤³¤Î¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤Î¤¿¤á¤ÎÀßÄê¤ò¼èÆÀ¤¹¤ë¤è¤¦¤Ë¤¹¤ë¤È ÊØÍø¤Ë¤Ê¤êÆÀ¤Þ¤¹¡£ ÄÌ¾ï DHCP ¥¯¥é¥¤¥¢¥ó¥È¤¬¥µ¥Ý¡¼¥È¤·¤Æ¤¤¤ë³Æ¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤Ï¡¢ ¤½¤Î¥ê¡¼¥¹¤ò³ÍÆÀ¤·´ÉÍý¤¹¤ë¤¿¤á¤Ë¡¢ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Î¾õÂÖµ¡³£¤ò¼Â¹Ô¤·¤Æ¤¤¤Þ¤¹¡£ ²¾ÁÛ¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤Ï¡¢*real-name* ¤ÈÌ¾ÉÕ¤±¤é¤ì¤¿¥¤¥ó¥¿¥Õ¥§¡¼¥¹¾å¤Ç ²ÔÆ¯¤·¤Æ¤¤¤ë¡¢¤Þ¤µ¤·¤¯¤â¤¦°ì¤Ä¤Î¾õÂÖµ¡³£¤Ç¤¹¡£ ¤³¤Îµ¡Ç½¤ò»ÈÍÑ¤¹¤ë¾ì¹ç¡¢ ²¾ÁÛ¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤È¼Âºß¤Î¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤ÎÎ¾Êý¤ËÂÐ¤·¤Æ ¥¯¥é¥¤¥¢¥ó¥È¼±ÊÌ»Ò¤òÄó¶¡¤·¤Ê¤±¤ì¤Ð¤Ê¤ê¤Þ¤»¤ó¡£ ¤Þ¤¿¡¢»ÈÍÑ¤·¤¿¤¤ IP ¥¢¥É¥ì¥¹¤ËÂÐ¤¹¤ë²¾ÁÛ¥¤¥ó¥¿¥Õ¥§¡¼¥¹ÍÑ¤Ë Ê¬Î¥¤µ¤ì¤¿¥¯¥é¥¤¥¢¥ó¥È¥¹¥¯¥ê¥×¥È¤òÄó¶¡¤·¤Ê¤±¤ì¤Ð¤Ê¤ê¤Þ¤»¤ó¡£ Îã¤¨¤Ð¼¡¤Î¤è¤¦¤Ë¤Ê¤ê¤Þ¤¹:

```
interface "ep0" {
        send dhcp-client-identifier "my-client-ep0";
}
pseudo "secondary" "ep0" {
        send dhcp-client-identifier "my-client-ep0-secondary";
        script "/etc/dhclient-secondary";
}
```

²¾ÁÛ¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤Î¤¿¤á¤Î¥¯¥é¥¤¥¢¥ó¥È¥¹¥¯¥ê¥×¥È¤Ï ¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤òÍ¸ú¤Ë¤·¤¿¤ê¶µ¸ú¤Ë¤·¤¿¤ê¤¹¤ëÀßÄê¤ò¤¹¤ë¤Ù¤¤Ç¤Ï¤¢¤ê¤Þ¤»¤ó¡£ ÆÃ¤Ë¡¢¥ê¡¼¥¹¤Î³ÍÆÀ¤ä¹¹¿·¤Î¾õÂÖ¡¢¤½¤·¤Æ¥ê¡¼¥¹¤Î´ü¸ÂÀÚ¤ì¤Î¾õÂÖ¤ò ¼è¤ê°·¤¦¤¿¤á¤Ë¤Ï¡¢¤½¤Î³¤È¤¬É¬Í×¤Ç¤¹¡£ ¾Ü°Ù¤Ï **dhclient-script(8)** ¤ò»²¾È¤·¤Æ²¼¤µ¤¤¡£

media "*media setup*" *[* , "*media setup*", *...]*;

media Ê¸¤Ï¡¢IP ¥¢¥É¥ì¥¹¼èÆÀÃæ¤Ë»ÈÍÑ¤¬»î¤ß¤é¤ì¤ë¡¢¥á¥Ç¥£¥¢ÀßÄê¥Ñ¥é¥á¡¼¥¿¤ò 1 ¤Ä°Ê¾åÄêµÁ¤·¤Þ¤¹¡£dhcp ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢¥ê¥¹¥ÈÃæ¤Î³Æ media setup Ê¸»úÎó¤ò ½ç¼¡»ÈÍÑ¤·¡¢¤¢¤ë¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤ò½¤ì¤ÇÀßÄê¤·¡¢¥Ö¡¼¥È¤ò»î¤ß¤Þ¤¹¡£ ÂÌÜ¤Ê¤é¤Ð¼¡¤Î media setup Ê¸»úÎó¤ò»ÈÍÑ¤·¤Þ¤¹¡£¤³¤ÎÊ¸¤Ï¡¢ ¥á¥Ç¥£¥¢¥¿¥¤¥×¤ò¸¡½Ð¤¹¤ëÇ½ÎÏ¤ò»ý¤¿¤Ê¤¤¥Í¥Ã¥È¥ï¡¼¥¯¥¤¥ó¥¿¥Õ¥§¡¼¥¹¤Ë ÂÐ¤·¤ÆÍøÍÑ¤Ç¤ ¤Þ¤¹¡£¥µ¡¼¥Ð¤Ø¤Î¥ê¥¯¥¨¥¹¥È¤¬¤Ç¤±þÅú¤¬ÆÀ¤é¤ì¤ë¤â¤Î ¤Ê¤é¤Ð¡¢¤É¤Î¤è¤¦¤Ê¥á¥Ç¥£¥¢¥¿¥¤¥×¤Ç¤â¤¿¤Ö¤óÀµÅö¤Ç¤¹ (ÊÝ¾Ú¤Ï¤·¤Þ¤»¤ó¤¬)¡£

media setup ¤Ï¥¢¥É¥ì¥¹¼èÆÀ¤Î½é´ü¥Õ¥§¡¼¥º (DHCPDISCOVER ¥Ñ¥±¥Ã¥È¤È DHCPOFFER ¥Ñ¥±¥Ã¥È)¤Ç¤Î¤ß»ÈÍÑ¤µ¤ì¤Þ¤¹¡£¤Ò¤È¤¿¤Ó¥¢¥É¥ì¥¹¤¬¼èÆÀ¤µ¤ì¤ë¤È¡¢ dhcp ¥¯¥é¥¤¥¢¥ó¥È¤Ï¤½¤Î¥¢¥É¥ì¥¹¤ò¥ê¡¼¥¹¥Ç¡¼¥¿¥Ù¡¼¥¹¤ËµÏ¿¤·¡¢ ¤½¤Î¥¢¥É¥ì¥¹¤òÆÀ¤ëºÝ¤ËÍÑ¤¤¤¿¥á¥Ç¥£¥¢¥¿¥¤¥×¤òµÏ¿¤·¤Þ¤¹¡£¥¯¥é¥¤¥¢¥ó¥È¤¬ ¥ê¡¼¥¹¤ò¹¹¿·¤·¤è¤¦¤È¤¹¤ëºÝ¤Ë¤Ï¾ï¤Ë¡¢¤½¤ì¤ÈÆ±¤¸¥á¥Ç¥£¥¢¥¿¥¤¥×¤ò»ÈÍÑ¤·¤Þ¤¹¡£ ¥ê¡¼¥¹¤ò´ü¸ÂÀÚ¤ì¤Ë¤·¤Æ¤Ï¤¸¤á¤Æ¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ï¥á¥Ç¥£¥¢¥¿¥¤¥×¤ò½ç¤Ë»î¤¹ ¾õÂÖ¤ËÌá¤ê¤Þ¤¹¡£

»ÈÍÑÎã

°Ê²¼¤ÎÀßÄê¥Õ¥¡¥¤¥ë¤Ï¡¢NetBSD 1.3 ¤ò¼Â¹Ô¤¹¤ë¤¢¤ë¥é¥Ã¥×¥È¥Ã¥×¥Þ¥·¥ó¤Ç »ÈÍÑ¤µ¤ì¤Æ¤¤¤ë¤â¤Î¤Ç¤¹¡£¤³¤Î¥Þ¥·¥ó¤Ï¡¢IP ¥¨¥¤¥ê¥¢¥¹¤È¤·¤Æ 192.5.5.213¡¢ ¥¤¥ó¥¿¥Õ¥§¡¼¥¹ ep0 (3Com 3C589C) ¤ò¤Ò¤È¤Ä»ý¤Ã¤Æ¤¤¤Þ¤¹¡£¤³¤Î¥¯¥é¥¤¥¢¥ó¥È ¤Ï¡¢DHCP ³èÆ°¤¬¤Û¤È¤ó¤É¤Ê¤¤¥Í¥Ã¥È¥ï¡¼¥¯¤Ç»þ´Ö¤ÎÂçÉôÊ¬¤ò¾ÃÈñ¤¹¤ë¤³¤È¤¬ ¤ï¤«¤Ã¤Æ¤¤¤ë¤Î¤Ç¡¢¥Ö¡¼¥È´Ö³Ö¤Ï¥Ç¥Õ¥©¥ë¥ÈÃÍ¤«¤é¤¤¤¯¤Ö¤ó¾®¤µ¤¯¤·¤Æ ¤¢¤ê¤Þ¤¹¡£¤³¤Î¥Þ¥·¥ó¤ÏÊ£¿ô¥Í¥Ã¥È¥ï¡¼¥¯´Ö¤Ç¥í¡¼¥ß¥ó¥° (°ÜÆ°) ¤·¤Þ¤¹¡£

```
timeout 60;
retry 60;
reboot 10;
select-timeout 5;
initial-interval 2;
reject 192.33.137.209;
```

```
interface "ep0" {
   send host-name "andare.fugue.com";
   send dhcp-client-identifier 1:0:a0:24:ab:fb:9c;
   send dhcp-lease-time 3600;
   supersede domain-name "fugue.com rc.vix.com home.vix.com";
   prepend domain-name-servers 127.0.0.1;
   request subnet-mask, broadcast-address, time-offset, routers,
           domain-name, domain-name-servers, host-name;
   require subnet-mask, domain-name-servers;
   script "CLIENTBINDIR/dhclient-script";
   media "media 10baseT/UTP", "media 10base2/BNC";
}

alias {
  interface "ep0";
  fixed-address 192.5.5.213;
  option subnet-mask 255.255.255.255;
}
```

¤³¤ì¤Ï dhclient.conf ¥Õ¥¡¥¤¥ë¤È¤·¤Æ¤ÏÈó¾ï¤ËÊ£»¨¤Ê¤â¤Î¤Ç¤¹¡£°ìÈÌ¤Ë¡¢³§¤µ¤ó¤¬»ÈÍÑ¤¹¤ë¤â¤Î¤Ï¤ë¤«¤Ë´ÊÃ±¤Ê¤Ï¤º¤Ç¤¹¡£Â¿¤¯¤Î¾ì¹ç¡¢dhclient.conf ¥Õ¥¡¥¤¥ë¤È¤·¤Æ¶õ¤Î¥Õ¥¡¥¤¥ë¤òÀ¸À®¤¹¤ë¤À±¤Ç½½Ê¬¤Ê¤Ï¤º¤Ç¤¹¡£ ¤Ä¤Þ¤ê¡¢¥Ç¥Õ¥©¥ë¥ÈÃÍ¤Ç¤è¤¤¤Î¤¬ÉáÄÌ¤Ç¤¹¡£

´ØÏ¢¹àÌÜ

dhcp-options(5), dhclient.leases(5), dhclient(8), RFC2132, RFC2131

ºî¼Ô

dhclient(8) ¤Ï Vixie Labs ¤È¤Î·ÀÌó¤Î¤â¤È¤Ç Ted Lemon ¤¬½ñ¤¤Þ¤·¤¿¡£ ËÜ¥×¥í¥¸¥§¥¯¥È¤Î´ð¶â¤Ï Internet Systems Consortium ¤¬Äó¶¡¤·¤Þ¤·¤¿¡£ Internet Systems Consortium ¤Ë´Ø¤¹¤ë¾ðÊó¤Ï¡¢ **http://www.isc.org** ¤Ë¤¢¤ê¤Þ¤¹¡£

NAME

dhclient.leases - DHCP client lease database

DESCRIPTION

The Internet Systems Consortium DHCP client keeps a persistent database of leases that it has acquired that are still valid. The database is a free-form ASCII file containing one valid declaration per lease. If more than one declaration appears for a given lease, the last one in the file is used. The file is written as a log, so this is not an unusual occurrence.

The format of the lease declarations is described in **dhclient.conf(5).**

FILES

/var/db/dhclient.leases

SEE ALSO

dhclient(8), dhcp-options(5), dhclient.conf(5), dhcpd(8), dhcpd.conf(5), RFC2132, RFC2131.

AUTHOR

dhclient(8) was written by Ted Lemon under a contract with Vixie Labs. Funding for this project was provided by Internet Systems Consortium. Information about Internet Systems Consortium can be found at **http://www.isc.org.**

Ì¾¾Î

dhclient.leases - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Î¥ê¡¼¥¹¥Ç¡¼¥¿¥Ù¡¼¥¹

²òÀâ

Internet Systems Consortium ¤Î DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢ ³ÍÆÀ¤·¤¿¥ê¡¼¥¹¤Î¡¦¤Á¤Þ¤ÀÍ-¸ú¤Ç¤¢¤ë¤â¤Î¤ò´ÉÍý¤¹¤ë¤¿¤á¤Î¡¢ ±ÊÂ³Åª¤Ê¥Ç¡¼¥¿¡¼¥¹¤òÊÝ»ý¤·¤Þ¤¹¡£ ¤³¤Î¥Ç¡¼¥¿¡¼¥¹¤Ï¡¢¼«Í³·Á¼°¤Î ASCII ¥Õ¥¡¥¤¥ë¤Ç¤¢¤ê¡¢ ¥ê¡¼¥¹ 1 ¤Ä¤Ë¤Ä¤Í¸ú¤ÊÀë¸À¤ò 1 ¤Ä´Þ¤ß¤Þ¤¹¡£ ¤¢¤ë¥ê¡¼¥¹¤ËÂÐ¤·¤ÆÊ£¿ô¤ÎÀë¸À¤¬ÅÐ¾ì¤¹¤ë¾ì¹ç¡¢ ¥Õ¥¡¥¤¥ëÃæ¤ÎºÇ¸å¤Î¤â¤Î¤¬»ÈÍÑ¤µ¤ì¤Þ¤¹¡£ ¤³¤Î¥Õ¥¡¥¤¥ë¤Ï¥í¥°¤È¤·¤Æ½ñ¤¹þ¤Þ¤ì¤Þ¤¹¤Î¤Ç¡¢ ¤³¤Î¤è¤¦¤Ê¾õÂÖ¤Ë¤Ê¤ë¤³¤È¤Ï°Û¾ï¤Ç¤Ï¤¢¤ê¤Þ¤»¤ó¡£

¥ê¡¼¥¹Àë¸À¤Î½ñ¼°¤Ï¡¢ **dhclient.conf(5)** ¤Ë½Ò¤µ¤ì¤Æ¤¤¤Þ¤¹¡£

´ØÏ¢¥Õ¥¡¥¤¥ë

DBDIR/dhclient.leases

´ØÏ¢¹àÌÜ

dhclient(8), dhcp-options(5), dhclient.conf(5), RFC2132, RFC2131

ºî¼Ô

dhclient(8) ¤Ï¡¢Vixie Labs ¤È¤Î·ÀÌó¤Î¤â¤È¤Ç¡¢Ted Lemon ¤¬µ½Ò¤·¤Þ¤·¤¿¡£ ËÜ¥×¥í¥¸¥§¥¯¥È¤Î»ñ¶â¤Ï¡¢Internet Systems Consortium ¤¬Äó¶¡¤·¤Þ¤·¤¿¡£ Internet Systems Consortium ¤Ë´Ø¤¹¤ë¾ðÊó¤Ï¡¢ **http://www.isc.org** ¤Ë¤¢¤ê¤Þ¤¹¡£

NAME

dhcp-eval - ISC DHCP conditional evaluation

DESCRIPTION

The Internet Systems Consortium DHCP client and server both provide the ability to perform conditional behavior depending on the contents of packets they receive. The syntax for specifying this conditional behaviour is documented here.

REFERENCE: CONDITIONAL BEHAVIOUR

Conditional behaviour is specified using the if statement and the else or elsif statements. A conditional statement can appear anywhere that a regular statement (e.g., an option statement) can appear, and can enclose one or more such statements. A typical conditional statement in a server might be:

```
if option dhcp-user-class = "accounting" {
  max-lease-time 17600;
  option domain-name "accounting.example.org";
  option domain-name-servers ns1.accounting.example.org,
                             ns2.accounting.example.org;
} elsif option dhcp-user-class = "sales" {
  max-lease-time 17600;
  option domain-name "sales.example.org";
  option domain-name-servers ns1.sales.example.org,
                             ns2.sales.example.org;
} elsif option dhcp-user-class = "engineering" {
  max-lease-time 17600;
  option domain-name "engineering.example.org";
  option domain-name-servers ns1.engineering.example.org,
                             ns2.engineering.example.org;
} else {
  max-lease-time 600;
  option domain-name "misc.example.org";
  option domain-name-servers ns1.misc.example.org,
                             ns2.misc.example.org;
}
```

On the client side, an example of conditional evaluation might be:

```
# example.org filters DNS at its firewall, so we have to use their DNS
# servers when we connect to their network.  If we are not at
# example.org, prefer our own DNS server.
if not option domain-name = "example.org" {
  prepend domain-name-servers 127.0.0.1;
}
```

The **if** statement and the **elsif** continuation statement both take boolean expressions as their arguments. That is, they take expressions that, when evaluated, produce a boolean result. If the expression evaluates to true, then the statements enclosed in braces following the **if** statement are executed, and all subsequent **elsif** and **else** clauses are skipped. Otherwise, each subsequent **elsif** clause's expression is checked, until an elsif clause is encountered whose test evaluates to true. If such a clause is found, the statements in braces following it are executed, and then any subsequent **elsif** and **else** clauses are skipped. If all the **if** and **elsif** clauses are checked but none of their expressions evaluate true, then if there is an **else** clause, the statements enclosed in braces following the **else** are evaluated. Boolean expressions that evaluate to null are treated as false in conditionals.

BOOLEAN EXPRESSIONS

The following is the current list of boolean expressions that are supported by the DHCP distribution.

data-expression-1 = *data-expression-2*

The = operator compares the values of two data expressions, returning true if they are the same, false if

they are not. If either the left-hand side or the right-hand side are null, the result is also null.

boolean-expression-1 **and** *boolean-expression-2*

The **and** operator evaluates to true if the boolean expression on the left-hand side and the boolean expression on the right-hand side both evaluate to true. Otherwise, it evaluates to false. If either the expression on the left-hand side or the expression on the right-hand side are null, the result is null.

boolean-expression-1 **or** *boolean-expression-2*

The **or** operator evaluates to true if either the boolean expression on the left-hand side or the boolean expression on the right-hand side evaluate to true. Otherwise, it evaluates to false. If either the expression on the left-hand side or the expression on the right-hand side are null, the result is null.

not *boolean-expression*

The **not** operator evaluates to true if *boolean-expression* evaluates to false, and returns false if *boolean-expression* evaluates to true. If *boolean-expression* evaluates to null, the result is also null.

exists *option-name*

The **exists** expression returns true if the specified option exists in the incoming DHCP packet being processed.

known

The **known** expression returns true if the client whose request is currently being processed is known - that is, if there's a host declaration for it.

static

The **static** expression returns true if the lease assigned to the client whose request is currently being processed is derived from a static address assignment.

DATA EXPRESSIONS

Several of the boolean expressions above depend on the results of evaluating data expressions. A list of these expressions is provided here.

substring (*data-expr***,** *offset***,** *length***)**

The **substring** operator evaluates the data expression and returns the substring of the result of that evaluation that starts *offset* bytes from the beginning, continuing for *length* bytes. *Offset* and *length* are both numeric expressions. If *data-expr*, *offset* or *length* evaluate to null, then the result is also null. If *offset* is greater than or equal to the length of the evaluated data, then a zero-length data string is returned. If *length* is greater than the remaining length of the evaluated data after *offset*, then a data string containing all data from *offset* to the end of the evaluated data is returned.

suffix (*data-expr***,** *length***)**

The **suffix** operator evaluates *data-expr* and returns the last *length* bytes of the result of that evaluation. *Length* is a numeric expression. If *data-expr* or *length* evaluate to null, then the result is also null. If *suffix* evaluates to a number greater than the length of the evaluated data, then the evaluated data is returned.

option *option-name*

The **option** operator returns the contents of the specified option in the packet to which the server is responding.

config-option *option-name*

The **config-option** operator returns the value for the specified option that the DHCP client or server has been configured to send.

hardware

The **hardware** operator returns a data string whose first element is the type of network interface indicated in packet being considered, and whose subsequent elements are client's link-layer address. If there is no packet, or if the RFC2131 *hlen* field is invalid, then the result is null. Hardware types

include ethernet (1), token-ring (6), and fddi (8). Hardware types are specified by the IETF, and details on how the type numbers are defined can be found in RFC2131 (in the ISC DHCP distribution, this is included in the doc/ subdirectory).

packet (*offset***,** *length***)**

The **packet** operator returns the specified portion of the packet being considered, or null in contexts where no packet is being considered. *Offset* and *length* are applied to the contents packet as in the **substring** operator.

string

A string, enclosed in quotes, may be specified as a data expression, and returns the text between the quotes, encoded in ASCII. The backslash ('\') character is treated specially, as in C programming: '\t' means TAB, '\r' means carriage return, '\n' means newline, and '\b' means bell. Any octal value can be specified with '\nnn', where nnn is any positive octal number less than 0400. Any hexadecimal value can be specified with '\xnn', where nn is any positive hexadecimal number less than or equal to 0xff.

colon-separated hexadecimal list

A list of hexadecimal octet values, separated by colons, may be specified as a data expression.

concat (*data-expr1***, ...,** *data-exprN***)**

The expressions are evaluated, and the results of each evaluation are concatenated in the sequence that the subexpressions are listed. If any subexpression evaluates to null, the result of the concatenation is null.

reverse (*numeric-expr1***,** *data-expr2***)**

The two expressions are evaluated, and then the result of evaluating the data expression is reversed in place, using hunks of the size specified in the numeric expression. For example, if the numeric expression evaluates to four, and the data expression evaluates to twelve bytes of data, then the reverse expression will evaluate to twelve bytes of data, consisting of the last four bytes of the the input data, followed by the middle four bytes, followed by the first four bytes.

leased-address

In any context where the client whose request is being processed has been assigned an IP address, this data expression returns that IP address.

binary-to-ascii (*numeric-expr1***,** *numeric-expr2***,** *data-expr1***,** *data-expr2***)**

Converts the result of evaluating data-expr2 into a text string containing one number for each element of the result of evaluating data-expr2. Each number is separated from the other by the result of evaluating data-expr1. The result of evaluating numeric-expr1 specifies the base (2 through 16) into which the numbers should be converted. The result of evaluating numeric-expr2 specifies the width in bits of each number, which may be either 8, 16 or 32.

As an example of the preceding three types of expressions, to produce the name of a PTR record for the IP address being assigned to a client, one could write the following expression:

```
concat (binary-to-ascii (10, 8, ".",
                         reverse (1, leased-address)),
        ".in-addr.arpa.");
```

encode-int (*numeric-expr***,** *width***)**

Numeric-expr is evaluated and encoded as a data string of the specified width, in network byte order (most significant byte first). If the numeric expression evaluates to the null value, the result is also null.

pick-first-value (*data-expr1* [*... expr*n] **)**

The pick-first-value function takes any number of data expressions as its arguments. Each expression is evaluated, starting with the first in the list, until an expression is found that does not evaluate to a null value. That expression is returned, and none of the subsequent expressions are evaluated. If all expressions evaluate to a null value, the null value is returned.

host-decl-name

The host-decl-name function returns the name of the host declaration that matched the client whose request is currently being processed, if any. If no host declaration matched, the result is the null value.

NUMERIC EXPRESSIONS

Numeric expressions are expressions that evaluate to an integer. In general, the maximum size of such an integer should not be assumed to be representable in fewer than 32 bits, but the precision of such integers may be more than 32 bits.

extract-int (*data-expr***,** *width***)**

The **extract-int** operator extracts an integer value in network byte order from the result of evaluating the specified data expression. Width is the width in bits of the integer to extract. Currently, the only supported widths are 8, 16 and 32. If the evaluation of the data expression doesn't provide sufficient bits to extract an integer of the specified size, the null value is returned.

lease-time

The duration of the current lease - that is, the difference between the current time and the time that the lease expires.

number

Any number between zero and the maximum representable size may be specified as a numeric expression.

client-state

The current state of the client instance being processed. This is only useful in DHCP client configuration files. Possible values are:

- Booting - DHCP client is in the INIT state, and does not yet have an IP address. The next message transmitted will be a DHCPDISCOVER, which will be broadcast.
- Reboot - DHCP client is in the INIT-REBOOT state. It has an IP address, but is not yet using it. The next message to be transmitted will be a DHCPREQUEST, which will be broadcast. If no response is heard, the client will bind to its address and move to the BOUND state.
- Select - DHCP client is in the SELECTING state - it has received at least one DHCPOFFER message, but is waiting to see if it may receive other DHCPOFFER messages from other servers. No messages are sent in the SELECTING state.
- Request - DHCP client is in the REQUESTING state - it has received at least one DHCPOFFER message, and has chosen which one it will request. The next message to be sent will be a DHCPREQUEST message, which will be broadcast.
- Bound - DHCP client is in the BOUND state - it has an IP address. No messages are transmitted in this state.
- Renew - DHCP client is in the RENEWING state - it has an IP address, and is trying to contact the server to renew it. The next message to be sent will be a DHCPREQUEST message, which will be unicast directly to the server.
- Rebind - DHCP client is in the REBINDING state - it has an IP address, and is trying to contact any server to renew it. The next message to be sent will be a DHCPREQUEST, which will be broadcast.

REFERENCE: LOGGING

Logging statements may be used to send information to the standard logging channels. A logging statement includes an optional priority (**fatal**, **error**, **info**, or **debug**), and a data expression.

log (*priority***,** *data-expr***)**

Logging statements take only a single data expression argument, so if you want to output multiple data values, you will need to use the **concat** operator to concatenate them.

REFERENCE: DYNAMIC DNS UPDATES

The DHCP client and server have the ability to dynamically update the Domain Name System. Within the configuration files, you can define how you want the Domain Name System to be updated. These updates are RFC 2136 compliant so any DNS server supporting RFC 2136 should be able to accept updates from the DHCP server.

SECURITY

Support for TSIG and DNSSEC is not yet available. When you set your DNS server up to allow updates from the DHCP server or client, you may be exposing it to unauthorized updates. To avoid this, the best you can do right now is to use IP address-based packet filtering to prevent unauthorized hosts from submitting update requests. Obviously, there is currently no way to provide security for client updates - this will require TSIG or DNSSEC, neither of which is yet available in the DHCP distribution.

Dynamic DNS (DDNS) updates are performed by using the **dns-update** expression. The **dns-update** expression is a boolean expression that takes four parameters. If the update succeeds, the result is true. If it fails, the result is false. The four parameters that the are the resource record type (RR), the left hand side of the RR, the right hand side of the RR and the ttl that should be applied to the record. The simplest example of the use of the function can be found in the reference section of the dhcpd.conf file, where events are described. In this example several statements are being used to make the arguments to the **dns-update**.

In the example, the first argument to the first Bdns-update expression is a data expression that evaluates to the A RR type. The second argument is constructed by concatenating the DHCP host-name option with a text string containing the local domain, in this case "ssd.example.net". The third argument is constructed by converting the address the client has been assigned from a 32-bit number into an ascii string with each byte separated by a ".". The fourth argument, the TTL, specifies the amount of time remaining in the lease (note that this isn't really correct, since the DNS server will pass this TTL out whenever a request comes in, even if that is only a few seconds before the lease expires).

If the first **dns-update** statement succeeds, it is followed up with a second update to install a PTR RR. The installation of a PTR record is similar to installing an A RR except that the left hand side of the record is the leased address, reversed, with ".in-addr.arpa" concatenated. The right hand side is the fully qualified domain name of the client to which the address is being leased.

SEE ALSO

dhcpd.conf(5), dhcpd.leases(5), dhclient.conf(5), dhcp-eval(5), dhcpd(8), dhclient(8), RFC2132, RFC2131.

AUTHOR

The Internet Systems Consortium DHCP Distribution was written by Ted Lemon under a contract with Vixie Labs. Funding for this project was provided through Internet Systems Consortium. Information about Internet Systems Consortium can be found at **http://www.isc.org.**

Ì¾¾Î

dhcp-eval - ISC DHCP ¤Ë¤ª¤±¤ë¾ò·ïÉÕ¤É¾²Á

²òÀâ

Internet Systems Consortium ¤Î DHCP ¥¯¥é¥¤¥¢¥ó¥È¤È¥µ¡¼¥Ð¤Ï¡¢¤É¤Á¤é¤â ¼õ¿®¤¤¤ë¥Ñ¥±¥Ã¥È¤Ë°ÍÂ¸¤·¤¿¾ò·ïÉÕ¤Æ°ºî¤¹Ô¤¦Ç½ÎÏ¤ò»ý¤Á¤Þ¤¤¡£ ¾ò·ïÉÕ¤-Æ°ºî¤ÎÊ¸Ë¡¤ò¤³¤³¤Ë¼¨¤·¤Þ¤¤¡£

»²¾È: ¾ò·ïÉÕ¤Æ°ºî

¾ò·ïÉÕ¤Æ°ºî¤Ï¡¢if, else, elsif Ê¸¤ò»ÈÍÑ¤·¤Æ»ØÄê¤·¤Þ¤¤¡£ ¾ò·ïÊ¸¤Ï¡¢ÄÌ¾ïÊ¸ (option Ê¸) ¤¬ÅÐ¾ì²ÄÇ½¤Ê¾ì½ê¤Ï¤É¤³¤Ë¤Ç¤âÅÐ¾ì²ÄÇ½¤Ç¤¢ê¡¢ ¤Þ¤¿¤³¤Î¤è¤¦¤ÊÊ¸¤ò³ç¤ë¤³¤È¤â²ÄÇ½¤Ç¤¹¤£ ¥µ¡¼¥Ð¤Ë¤ª¤±¤ë¾ò·ïÊ¸¤Ï¡¼¤Î¤è¤¦¤Ë¤Ê¤ë¤³¤È¤¬Â¿¤¤¤Ç¤·¤ç¤¦:

```
if option dhcp-user-class = "accounting" {
  max-lease-time 17600;
  option domain-name "accounting.example.org";
  option domain-name-servers ns1.accounting.example.org,
                             ns2.accounting.example.org;
} elsif option dhcp-user-class = "sales" {
  max-lease-time 17600;
  option domain-name "sales.example.org";
  option domain-name-servers ns1.sales.example.org,
                             ns2.sales.example.org;
} elsif option dhcp-user-class = "engineering" {
  max-lease-time 17600;
  option domain-name "engineering.example.org";
  option domain-name-servers ns1.engineering.example.org,
                             ns2.engineering.example.org;
} else {
  max-lease-time 600;
  option domain-name "misc.example.org";
  option domain-name-servers ns1.misc.example.org,
                             ns2.misc.example.org;
}
```

¥¯¥é¥¤¥¢¥ó¥ÈÂ¦¤Ç¤Ï¡¢¾ò·ïÉÕ¤É¾²Á¤ÎÎã¤Ï¼¡¤Î¤è¤¦¤Ë¤Ê¤ë¤Ç¤·¤ç¤¦:

```
# example.org ¤Ï¥Õ¥¡¥¤¥ä¥¦¥©¡¼¥ë¤Ç DNS ¤ò¥Õ¥£¥ë¥¿¤¹¤ë¤Î¤Ç¡¢
# example.org ¥Í¥Ã¥È¥ï¡¼¥¯¯¤Ë·Ò¤¬¤ë¤È¤¤Î¤ß¡¢¤½¤Î DNS ¥µ¡¼¥Ð¤ò»ÈÍÑ¤·¤Þ¤¹¡£
# example.org ¤Ë·Ò¤¬¤ë¤Î¤Ç¤Ï¤Ê¤¤¾ì¹ç¡¢¼«¸Ê¤Î DNS ¥µ¡¼¥Ð¤òÍ¥ÀèºÈÍÑ¤·¤Þ¤¹¡£
if not option domain-name = "example.org" {
  prepend domain-name-servers 127.0.0.1;
}
```

if Ê¸¤È **elsif** ·ÑÂ³Ê¸¤Ï¡¢°ú¿ô¤È¤·¤Æ¥Ö¡¼¥ë¼°¤ò¼è¤ê¤Þ¤¹¡£ ¤Ä¤Þ¤ê¡¢¤³¤ì¤é¤ÎÊ¸¤Ï¡¢É¾²Á¤µ¤ì¤ë¤È¥Ö¡¼¥ëÃÍ¤Î·ë²Ì¤òÀ¸À®¤¹¤ë¼°¤ò¼è¤ê¤Þ¤¹¡£ ¼°¤ÎÉ¾²Á·ë²Ì¤¬¿¿¤Ë¤Ê¤ë¤È¡¢ **if** Ê¸¤ÎÄ¾¸å¤Î¥Ö¡¼¥¹¤Ç³¤é¤ì¤¿Ê¸¤é¤Â¹Ô¤µ¤ì¡¢¸å³¤¹¤ë **elsif** ¤È **else** ¤ÎÀá¤Ï¥¥Ã¥×¤µ¤ì¤Þ¤¹¡£ ¤½¤¦¤Ç¤Ê¤¤¾ì¹ç¡¢É¾²Á·ë²Ì¤¬¿¿¤Ë¤Ê¤ë elsif Àá¤Ë½Ð²ñ¤¦¤Þ¤Ç¡¢¸å³¤¹¤ë³Æ **elsif** Àá¤Î¼°¤¬¥Á¥§¥Ã¥¯¤µ¤ì¤Þ¤¹¡£ ¤½¤Î¤è¤¦¤ÊÀá¤¬¸«ÉÕ¤«¤ë¤È¡¢Ä¾¸å¤Î¥Ö¡¼¥¹¤Æ¤ÎÊ¸¤¬¼Â¹Ô¤µ¤ì¡¢¸å³¤¹¤ë **elsif** ¤È **else** ¤ÎÀá¤Ï¥¥-¥Ã¥×¤µ¤ì¤Þ¤¹¡£ ¤¹¤Ù¤Æ¤Î **if** ¤ªº¤è¤Ó **elsif** ¤ÎÀá¤¬¥Á¥§¥Ã¥¯¤µ¤ì¤¿¸å¿¿¤Î¤Î¤É¤Î¼°¤â¿¿¤Ç¡¢¤Ë¤Ê¤é¤Ê¤¤¾ì¹ç¡¢ **else** Àá¤¬Â¸ºß¤¹¤ë¾ì¹ç¡¢ **else** ¤ÎÄ¾¸å¤Î¥Ö¡¼¥¹¤Æ¤ÎÊ¸¤¬É¾²Á¤µ¤ì¤Þ¤¹¡£ ¾ò·ï¤Ë¤ª¤¤¤Æ¤Ï¡¢É¾²Á·ë²Ì¤¬¶õ¤Ë¤Ê¤ë¥Ö¡¼¥ë¼°¤Ïµ¶¤È¤·¤Æ°·¤ï¤ì¤Þ¤¹¡£

¥Ö¡¼¥ë¼°

°Ê²¼¤Ï¡¢DHCP ÇÛÉÛÊª¤Ç¸½ºß¥µ¥Ý¡¼¥È¤µ¤Æ¤¤¤ë¥Ö¡¼¥ë¼°¤Î°ìÍ÷¤Ç¤¹¡£

data-expression-1 = data-expression-2

= ¥ª¥Ú¥ì¡¼¥¿¤Ï¡¢2 ¸Ä¤Î¥Ç¡¼¥¿¤òÈæ³Ó¤·¡¢Î¾¼Ô¤¬Æ±¤¸¾ì¹ç¤Ï¿¿¤òÊÖ¤·¡¢ Æ±°ì¤Ç¤Ê¤¤¾ì¹ç¤Ïµ¶¤òÊÖ¤·¤Þ¤¹¡£ º¸ÊÕ¤â¤·¤¯¯¤Ï±¦ÊÕ¤Î¤¤¤º¤ì¤«¤¬¶õ¤Î¾ì¹ç¡·ë²Ì¤Ï¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£

boolean-expression-1 **and** *boolean-expression-2*

and ¥ª¥Ú¥ì¡¼¥¿¤Ï¡¢º¸ÊÕ¤Î¥Ö¡¼¥ë¼°¤È±¦ÊÕ¤Î¥Ö¡¼¥ë¼°¤ÎÎ¾Êý¤ÎÉ¾²Á·ë²Ì¤¬ ¿¿¤Î¾ì¹ç¡¢¿¿¤ÈÉ¾²Á¤µ¤ì¤Þ¤¹¡£ ¤½¤¦¤Ç¤Ê¤¤¾ì¹ç¡µ¶¤ÈÉ¾²Á¤µ¤ì¤Þ¤¹¡£ º¸ÊÕ¤â¤·¤¯¯¤Ï±¦ÊÕ¤Î¤¤¤º¤ì¤«¤¬¶õ¤Î¾ì¹ç¡·ë²Ì¤Ï¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£

boolean-expression-1 **or** *boolean-expression-2*

or ¥ª¥Ú¥ì¡¼¥¿¤Ï¡¢º¸ÊÕ¤Î¥Ö¡¼¥ë¼°¤È±¦ÊÕ¤Î¥Ö¡¼¥ë¼°¤Î¤¤¤º¤ì¤«¤ÎÉ¾²Á·ë²Ì¤¬ ¿¿¤Î¾ì¹ç¡¢¿¿¤ÈÉ¾²Á¤µ¤ì¤Þ¤¹¡£ ¤½¤¦¤Ç¤Ê¤¤¾ì¹ç¡µ¶¤ÈÉ¾²Á¤µ¤ì¤Þ¤¹¡£ º¸ÊÕ¤â¤·¤¯¯¤Ï±¦ÊÕ¤Î¤¤¤º¤ì¤«¤¬¶õ¤Î¾ì¹ç¡·ë²Ì¤Ï¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£

not *boolean-expression*

not ¥ª¥Ú¥ì¡¼¥¿¤Ï¡¢*boolean-expression* ¤ÎÉ¾²Á·ë²Ì¤¬µ¶¤Î¾ì¹ç¡¢ ¿¿¤ÈÉ¾²Á¤µ¤ì¤Þ¤¹¡£ ¤Þ¤¿¡¢*boolean-expression* ¤ÎÉ¾²Á·ë²Ì¤¬¿¿¤Î¾ì¹ç¡µ¶¤ÈÉ¾²Á¤µ¤ì¤Þ¤¹¡£ *boolean-expression* ¤ÎÉ¾²Á·ë²Ì¤¬¶õ¤Î¾ì¹ç¡·ë²Ì¤â¤Þ¤¿¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£

exists *option-name*

exists ¼°¤Ï¡¢½èÍýÂÐ¾Ý¤ÎÆþÎÏ DCHP ¥Ñ¥±¥Ã¥ÈÃæ¤Ë¡¢ »ØÄê¤µ¤ì¤¿¥ª¥×¥·¥ç¥ó¤¬Â¸ºß¤¹¤ë¾ì¹ç¡¢¿¿¤òÊÖ¤·¤Þ¤¹¡£

known

known ¼°¤Ï¡¢Í×µáÂÐ±þÃæ¤Î¯¯¥é¥¤¥¢¥ó¥È¤¬´ûÃÎ¤Î¾ì¹ç¡¢ ¤¹¤Ê¤ï¤Á¥Û¥¹¥È¥Èë¸À¤¬¤¢¤ë¾ì¹ç¡¢¿¿¤òÊÖ¤·¤Þ¤¹¡£

static

static ¼°¤Ï¡¢Í×µáÂÐ±þÃæ¤Î¯¯¥é¥¤¥¢¥ó¥È¤Ø¤Î¥ê¡¼¥¹ä¤êÅö¤Æ¤¬¡¢ ÀÅÅª¥¢¥É¥¹¥¹ä¤êÅö¤Æ¤Ë¤è¤ë¤â¤Î¤Ç¤¢¤Ã¤¿¾ì¹ç¡¢¿¿¤òÊÖ¤·¤Þ¤¹¡£

¥Ç¡¼¥¿¿¼°

Á°½Ò¤Î¥Ö¡¼¥ë¼°¤Ï¡¢¥Ç¡¼¥¿¿¼°¤ÎÉ¾²Á·ë²Ì¤Ë°ÍÂ¸·¤Þ¤¹¡£ ¥Ç¡¼¥¿¿¼°¤ò¤³¤³¤Ë¼¨¤·¤Þ¤¹¡£

substring (*data-expr***,** *offset***,** *length***)**

substring ¥ª¥Ú¥ì¡¼¥Ï¡¢¥Ç¡¼¥¿¿¼°¤òÉ¾²Á¤·¡¢ É¾²Á·ë²ÌÃæ¤Î *offset* ¥Ð¥¤¥È¤«¤é³«»Ï¤·¤Æ *length* ¥Ð¥¤¥È·ÑÂ³¤¤ë ¥µ¥Ö¥¹¥È¥ê¥ó¥°¤òÊÖ¤·¤Þ¤¹¡£*offset* ¤È *length* ¤Ï¶¦¤Ë¿ôÃÍ¼°¤Ç¤¤¡£ *data-expr*, *offset*, *length* ¤Î¤¤º¤ì¤«¤¬¶õ¤È¤Ê¤Ã¤Æ¤¤¤ë¾ì¹ç¡¢ ·ë²Ì¤â¤Þ¤¿¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£ *offset* ¤¬¡¢É¾²Á¤µ¤ì¤¿¥Ç¡¼¥¿¤ÎÄ¹¤µ°Ê¾å¤Ç¤¢¤ë¾ì¹ç¡¢ Ä¹¤µ 0 ¤Î¥Ç¡¼¥¿¡¢»úÎó¤¬ÊÖ¤µ¤ì¤Þ¤¹¡£*length* ¤¬¡¢É¾²Á¤µ¤ì¤¿¥Ç¡¼¥¿¤Î *offset* ¤è¤ê¸å¤ÎÄ¹¤µ¤è¤êÂç¤¤¤¾ì¹ç¡¢ É¾²Á¤µ¤ì¤¿¥Ç¡¼¥¿¤Î *offset* ¤«¤é½ªÃ¼¤Þ¤Ç¤ÎÁ´¥Ç¡¼¥¿¡¢¤ò´Þ¤à ¥Ç¡¼¥¿¡¢»úÎó¤¬ÊÖ¤µ¤ì¤Þ¤¹¡£

suffix (*data-expr***,** *length***)**

suffix ¥ª¥Ú¥ì¡¼¥Ï¡¢*data-expr* ¤òÉ¾²Á¤·¡¢ É¾²Á·ë²Ì¤ÎºÇ¸å¤Î *length* ¥Ð¥¤¥È¤òÊÖ¤·¤Þ¤¹¡£ *length* ¤Ï¿ôÃÍ¼°¤Ç¤¤¡£ *data-expr* ¤Þ¤¿¤Ï *length* ¤ÎÉ¾²Á·ë²Ì¤¬¶õ¤Î¾ì¹ç¡¢ ·ë²Ì¤â¤Þ¤¿¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£ *suffix* (ÌõÃí: *length* ¤¬Àµ¤·¤¤¤È»×¤¤ï¤ì¤Þ¤¹) ¤ÎÉ¾²Á·ë²Ì¤¬É¾²Á¤µ¤ì¤¿¥Ç¡¼¥¿¤ÎÄ¹¤µ¤è¤êÂç¤¤¤¾ì¹ç¡¢ É¾²Á¤µ¤ì¤¿¥Ç¡¼¥¿¤¬ÊÖ¤µ¤ì¤Þ¤¹¡£

option *option-name*

option ¥ª¥Ú¥ì¡¼¥Ï¡¢¥µ¡¼¥Ð¤¬±þÅú½èÍýÃæ¤Î¥Ñ¥±¥Ã¥È¤ÎÃæ¤Î¡¢ »ØÄê¤·¤¿¥ª¥×¥·¥ç¥ó¤ÎÆâÍÆ¤òÊÖ¤·¤Þ¤¹¡£

config-option *option-name*

config-option ¥ª¥Ú¥ì¡¼¥Ï¡¢»ØÄê¤·¤¿¥ª¥×¥·¥ç¥ó¤ËÂÐ¤·¡¢ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Þ¤¿¤Ï¥µ¡¼¥Ð¤¬¥Á÷½Ð¤¤¤ë¤è¡¢ÀßÄê¤µ¤ì¤¿ÃÍ¤òÊÖ¤·¤Þ¤¹¡£

hardware

hardware ¥ª¥Ú¥ì¡¼¥Ï¡¢¥Ç¡¼¥¿¥¹¥È¥ê¥ó¥°¤òÊÖ¤·¤Þ¤¹¡£ ¥Ç¡¼¥¿¥¹¥È¥ê¥ó¥°¤ÎºÇ½é¤ÎÍ×ÁÇ¤Ï¡¢ ÂÐ¾Ý¥Ñ¥±¥Ã¥È¤¬¼¨¤¹¥¿¥¤¥×¤Ç¤¢¤ê¡¢¤½¤ÎÃÍ¤Ï¥¯¥é¥¤¥¢¥ó¥È¤¬¥µ¡¼¥Ð¤¹¤ë¥Ï¡¼¥É¥¦¥§¥¢¤Î¼ïÎà¤Ç¤¢¤ê¡¢ ¥Ï¡¼¥É¥¦¥§¥¢¤Î¥¢¥É¥ì¥¹¤Ç¤¹¡£

¸åÂ³¤¹¤ëÍ×ÁÇ¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤Î¥ê¥ó¥¯ÁØ¥¢¥É¥ì¥¹¤Ç¤¹¡£ ¥Ñ¥±¥Ã¥È¤¬Â¸ºß¤·¤Ê¤¤¾ì¹ç¤â¤·¤¯¯¤Ï RFC2131 *hlen* ¥Õ¥£¡¼¥ë¥É¤¬Ìµ¸ú¤Ê¾ì¹ç¡¢ ·ë²Ì¤Ï¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£ ¥Ï¡¼¥É¥¦¥§¢¥¿¥¤¥×¤Ë¤Ï¡¢¥¡¼¥µ¥Í¥Ã¥È (1)¡¢¥È¡¼¯¥ó¥ê¥ó¥° (6)¡¢ FDDI (8) ¤¬´Þ¤Þ¤ì¤Þ¤¹¡£ ¥Ï¡¼¥É¥¦¥§¢¥¿¥¤¥×¤Ï IETF ¤Ë¤è¤Ã¤Æµ¬Äê¤µ¤ì¡¢ ¤É¤Î¤è¦¤Ë¥¿¥¤¥×¤Î¿ôÃÍ¤¬ÄêµÁ¤µ¤ì¤ë¤«¾¾ÜºÙ¤Ï RFC2131 (ISC DHCP ÇÛÉÛÊª¤Ç¤Ï¡¢doc/¥µ¥Ö¥Ç¥£¥ì¯¯¥È¥ê¤Ë¤¢¤ê¤Þ¤¹) ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

packet (*offset***,** *length***)**

packet ¥ª¥Ú¥ì¡¼¥¿¤Ï¡¢ÂÐ¾Ý¥Ñ¥±¥Ã¥È¤Î»ØÄêÉôÊ¬¤òÊÖ¤¹¤«¡¢ ÂÐ¾Ý¥Ñ¥±¥Ã¥È¤¬Ìµ¤¤Ê¸Ì®¤Ç¤Ï¶õ¤òÊÖ¤·¤Þ¤¹¡£ *offset* ¤È *length* ¤Ï¡¢ **substring** ¥ª¥Ú¥ì¡¼¥¿¤È±ÍÍ¤Ë¡¢¥Ñ¥±¥Ã¥È¤ÎÆâÍÆ¤ËÅ¬ÍÑ¤µ¤ì¤Þ¤¹¡£

string

¥¯¥©¡¼¥È¤Ç³ç¤é¤ì¤¿¥¹¥È¥ê¥ó¥°¤Ï¥Ç¡¼¥¿¡¼°¤È¤·¤Æ»ØÄê²ÄÇ½¤Ç¤¤ê¡¢ ¥¯¥©¡¼¥È¤Î´Ö¤ò ASCII ¥¨¥ó¥³¡¼¥É¤·¤¿¤Î¥Æ¥¥¹¥È¤òÊÖ¤·¤Þ¤¹¡£ ¥Ð¥Ã¥¯¥¹¥é¥Ã¥·¥å (’\’) Ê¸»ú¤Ï C ¥×¥í¥°¥é¥à¤Î¤è¦¤ËÆÃÊÌ°·¤¤¤µ¤ì¤Þ¤¹: ¤¹¤Ê¤ï¤Á ’\t’ ¤Ï¥¿¥Ö¤ò¡¢’\r’ ¤ÏÉü²þ¹Ô¤ò¡¢’\n’ ¤Ï²þ¹Ô¤ò¡¢’\b’ ¤Ï¥Ù¥ë¤ò °ÕÌ£¤·¤Þ¤¹¡£ 8 ¿Ê¿ôÃÍ¤Ï ’\nnn’ ¤Ç»ØÄê²ÄÇ½¤Ç¤¤ê¡¢nnn ¤Ï 0 °Ê¾å 0377 °Ê²¼Î 8 ¿Ê¿ôÃÍ¤Ç¤¹¡£ 16 ¿Ê¿ôÃÍ¤Ï ’\xnn’ ¤Ç»ØÄê²ÄÇ½¤Ç¤¤ê¡¢nn ¤Ï 0 °Ê¾å 0xff °Ê²¼Î 16 ¿Ê¿ôÃÍ¤Ç¤¹¡£

colon-seperated hexadecimal list

¥³¥í¥ó¤Ç¶èÀÚ¤é¤ì¤¿ 16 ¿Ê¿ô¤Î¥ª¥¯¥Æ¥Ã¥ÈÃÍ¤Î¥ê¥¹¥È¤ò¡¢ ¥Ç¡¼¥¿¡¼°¤È¤·¤Æ»ØÄê²ÄÇ½¤Ç¤¹¡£

concat (*data-expr1***, ...,** *data-exprN***)**

¼°¤¬É¾²Á¤µ¤ì¡¢³ÆÉ¾²Á·ë²Ì¤¬µ¥Ö¼°¤Î½çÈÖ¤ËÏ¢·ë¤µ¤ì¤Þ¤¹¡£
¥µ¥Ö¼°¤Î¤¤°¤ì¤«¤ÎÉ¾²Á·ë²Ì¤¬¶õ¤Ë¤Ê¤ë¾ì¹ç¡¢Ï¢·ë¤Î·ë²Ì¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£

reverse (*numeric-expr1***,** *data-expr2***)**

2 ¸Ä¤Î¼°¤¬É¾²Á¤µ¤ì¡¢¥Ç¡¼¥¿¼°¤ÎÉ¾²Á·ë²Ì¤¬¤½¤Î¾ì¤ÇÈ¿Å¾¤µ¤ì¤Þ¤¹¡£ È¿Å¾¤Ï¡¢ôÃÍ¼°¤Ç»ØÄê¤µ¤ëÂç¤µ¤ÎÃ±°Ì¤Ç¹Ô¤ï¤ì¤Þ¤¹¡£ Îã¨¤Ð¡¢¿ôÃÍ¼°¤ÎÉ¾²Á·ë²Ì¤¬ 4 ¤Î¾ì¹ç¤Ç¡¢ ¥Ç¡¼¥¿¼°¤ÎÉ¾²Á·ë²Ì¤¬ 12 ¥Ð¥¤¥È¤Ë¤Ê¤ë¾ì¹ç¡¢ reverse ¼°¤ÎÉ¾²Á·ë²Ì¤Ï¡¼¡¢¤Î¤è¦¤Ê 12 ¥Ð¥¤¥È¤Î¥Ç¡¼¥¿¿¤Ë¤Ê¤ê¤Þ¤¹¡£ ¤¹¤Ê¤ï¤Á¡¢ÆþÎï¤Î°Ç¸å¤Î 4 ¥Ð¥¤¥È¡¢¿¿Ãæ¤Î 4¥Ð¥¤¥È¡¢°Ç½é¤Î 4 ¥Ð¥¤¥È¤Î ½ç¤Ë¤Ê¤ê¤Þ¤¹¡£

leased-address

¤¤¤«¤Ê¤ëÊ¸Ì®¤Ë¤ª¤¤¤Æ¤â¡¢ Í×µá½èÍýÂÐ¾Ý¤È¤Ê¤Ã¤Æ¤¤¤ë¥¯¥é¥¤¥¢¥ó¥È¤Ë IP ¥¢¥É¥ì¥¹¤¬³ä¤êÅö¤Æ°Ñ¤Î¾ì¹ç¡¢ ¤½¤Î IP ¥¢¥É¥ì¥¹¤¬ÊÖ¤µ¤ì¤Þ¤¹¡£

binary-to-ascii (*numeric-expr1***,** *numeric-expr2***,** *data-expr1***,** *data-expr2***)**

data-expr2 ¤ÎÉ¾²Á·ë²Ì¤ò¥Æ¥¥¹¥È¥¹¥È¥ê¥ó¥°¤ËÊÑ´¹¤·¤Þ¤¹¡£ ¤³¤Î¥Æ¥¥¹¥È¥¹¥È¥ê¥ó¥°¤Ãæ¤Ç¤Ï¡¢ data-expr2 ¤ÎÉ¾²Á·ë²Ì¤Î³ÆÍ×ÁÇ¤¬¡¢1 ¸Ä¤Î¿ôÃÍ¤Ë¤Ê¤ê¤Þ¤¹¡£ ³Æ¿ôÃÍ¤Ï¡¢¤½¤ì¤¾¤ì¡¢data-expr1 ¤ÎÉ¾²Á·ë²Ì¤Ë¤è¤Ã¤Æ¶èÀÚ¤é¤ì¤Þ¤¹¡£ numeric-expr1 ¤ÎÉ¾²Á·ë²Ì¤Ï¡¢´ð¿ô (2 ¤«¤é 16) ¤Ç¤¤ê¡¢ ¤³¤Î´ð¿ôÃÍ¤¬ÊÑ´¹¤µ¤ì¤Þ¤¹¡£ numeric-expr2 ¤ÎÉ¾²Á·ë²Ì¤Ï¡¢³Æ¿ôÃÍ¤Î¥Ó¥Ã¥ÈÉý¤Ç¤¤ê¡¢ 8, 16, 32 ¤Î¤¤°¤ì¤«¤Ç¤¹¡£

°Ç½é¤Î 3 ¸Ä¤Î¥¤¥×¤Î¼°¤ÎÎã¤È¤·¤Æ¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤Ë³ä¤êÅö¤Æ¤é¤ì¿ IP ¥¢¥É¥ì¥¹ÍÑ¤Î PTR ¥ì¥³¡¼¥É¤ÎÌ¾Á°¤òÀ¸À®¤¹¤ë¿¤á¤Ë»ÈÍ²ÄÇ½¤Ê¼°¤ò¼¨¤·¤Þ¤¹

```
concat (binary-to-ascii (10, 8, ".",
                         reverse (1, leased-address)),
        ".in-addr.arpa.");
```

encode-int (*numeric-expr***,** *width***)**

¿ôÃÍ¼°¤¬É¾²Á¤µ¤ì¡¢ØÄê¤µ¤ì¤¿Éý¤ÎÇ¡¼¥¹¥È¥ê¥ó¥°¤Ë ¥Í¥Ã¥È¥ï¡¼¯¥Ð¥¤¥È½ç (°Ç¾å°Ì¥Ð¥¤¥È¤¬°Ç½é) ¤Ç¥¨¥ó¥³¡¼¥É¤µ¤ì¤Þ¤¹¡£
¿ôÃÍ¼°¤ÎÉ¾²Á·ë²Ì¤¬¶õ¤ÎÃÍ¤Ë¤Ê¤ë¾ì¹ç¡¢ë²Ì¤â¤Þ¤¿¶õ¤Ç¤¹¡£

pick-first-value (*data-expr1* [*... exprn*] **)**

pick-first-value ´Ø¿ô¤Ï¡¢Ç¤°Õ¸Ä¤Î¥Ç¡¼¥¿¼°¤ò¼è¤êÆÀ¤Þ¤¹¡£

¥ê¥¹¥È¤ÎÀèÆ¬¤«¤é³Æ¼°¤¬É¾²Á¤µ¤ì¡¢ É¾²Á·ë²Ì¤¬¶õ¤Ç¤Ï¤Ê¤¤¼°¤¬¸«ÉÕ¤«¤ë¤Þ¤Ç¤³¤ì¤¬Â³¤-¤Þ¤¹¡£ ¤³¤Î¼°¤¬ÊÖ¤µ¤ì¡¢¤³¤Î¼°¤Ë¸åÂ³¤¹¤ë¼°¤ÏÉ¾²Á¤µ¤ì¤Þ¤»¤ó¡£ ¤¹¤Ù¤Æ¤Î¼°¤ÎÉ¾²Á·ë²Ì¤¬¶õ¤Î¾ì¹ç¡¢¶õ¤ÎÃÍ¤¬ÊÖ¤µ¤ì¤Þ¤¹¡£

host-decl-name

host-decl-name ´Ø¿ô¤Ï¡¢½ºßÍ×µá½èÍýÂÐ¾Ý¤È¤Ê¤Ã¤Æ¤¤¤ë¥¯¥é¥¤¥¢¥ó¥È¤Ë¥Þ¥Ã¥Á¤¹¤ë¡¢ ¥Û¥¹¥ÈÀë¸À¤ÎÌ¾Á°¤òÊÖ¤·¤Þ¤¹¡£ ¤É¤Î¥Û¥¹¥ÈÀë¸À¤â¥Þ¥Ã¥Á¤·¤Ê¤¤¾ì¹ç¡¢¤ë²Ì¤Ï¶õ¤Ë¤Ê¤ê¤Þ¤¹¡£

¿ôÃÍ¼°

¿ôÃÍ¼°¤Ï¡¢É¾²Á·ë²Ì¤¬À°¿ô¤Ë¤Ê¤ë¼°¤Ç¤¹¡£ °ìÈÌ¤Ë¡¢À°¿ô¤ÎºÇÂç¥µ¥¤¥º¤¬ 32 ¥Ó¥Ã¥È¤Ì¤Ëþ¤Ç¤¢¤ëÈ²¾Äê¤¹¤Ù¤¤Ç¤Ï¤¢¤ê¤Þ¤»¤ó¤¬¡¢ À°¿ô¤ÎÀºÅÙ¤¬ 32 ¥Ó¥Ã¥È¤ò±Û¤¨¤ë¤³¤È¤Ï¤¢¤êÆÀ¤Þ¤¹¡£

extract-int (*data-expr***, *width*)**

extract-int ¥ª¥Ú¥ì¼¥¿¤Ï¡¢¥Í¥Ã¥È¥ï¼¥¯¥Ð¥¤¥È½ç¤ÎÀ°¿ô¤ò¡¢ »ØÄê¤·¤¿¥Ç¡¼¥¿¼°¤ÎÉ¾²Á·ë²Ì¤«¤é¼è¤ê½Ð¤·¤Þ¤¹¡£ Éý¤Ï¡¢¼è¤ê½Ð¤¹À°¿ô¤Î¥Ó¥Ã¥ÈÉý¤Ç¤¹¡£ ¸½ºßß¡¢¥µ¥Ý¡¼¥È¤µ¤ì¤Æ¤¤¤ëÉý¤Ï 8, 16, 32 ¤Î¤¤º¤ì¤«¤Ç¤¹¡£ ¥Ç¡¼¥¿¼°¤ÎÉ¾²Á·ë²Ì¤¬¡¢»ØÄê¤·¤¿Âç¤µ¤ÎÀ°¿ô¤Ï¼è¤ê¤ê½Ð¤¤Î¤Ë ½½Ê¬¤Ê¥Ó¥Ã¥È¤òÄó¶¡¤·¤Ê¤¤¾ì¹ç¡¢¶õ¤ÎÃÍ¤¬ÊÖ¤µ¤ì¤Þ¤¹¡£

lease-time

¸½ºßß¤Î¥ê¡¼¥¹¤Î´ü´Ö¤Ç¤¹¡£ ¤¹¤Ê¤ï¤Á¡¢½ºßß¤Î»þ¹ï¤È¥ê¡¼¥¹¤Î´ü¸Â¤¬ÀÚ¤ì¤ë»þ¹ï¤È¤Î°¹¤Ç¤¹¡£

number

0 ¤«¤éÉ½¸½²ÄÇ½¤Ê°ÇÂç¥µ¥¤¥º¤ÎÈÏ°Ï¤ÎÇ¤°Õ¤Î¿ôÃÍ¤ò¡¢¿ôÃÍ¼°¤È¤·¤Æ»ØÄê²ÄÇ½¤Ç¤¹¡£

client-state

½èÍýÂÐ¾Ý¤Î¥¯¥é¥¤¥¢¥ó¥È¤Î¸½ºßß¤Î¾õÂÖ¤Ç¤¹¡£ DHCP ¥¯¥é¥¤¥¢¥ó¥ÈÀßÄê¥Õ¥¡¥¤¥ë¤Ë¤ª¤¤¤Æ¤Î¤ßÍÍÑ¤Ç¤¹¡£ ¼è¤êÆÀ¤ëÃÍ¤Ï¼¡¤ÎÄÌ¤ê¤Ç¤¹:

- Booting - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï INIT ¾õÂÖ¤Ç¤¢¤ê¡ IP ¥¢¥É¥ì¥¹¤ò¤Þ¤À»ý¤Á¤Þ¤»¤ó¡£ ¼¡¤ËÁ÷¿®¤µ¤ì¤ë¥á¥Ã¥»¡¼¥¸¤Ï DHCPDISCOVER ¤Ç¤¢¤ê¡¢ ¤³¤ì¤Ï¥Ö¥í¡¼¥É¥¥ã¥¹¥È¤µ¤ì¤Þ¤¹¡£
- Reboot - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï INIT-REBOOT ¾õÂÖ¤Ç¤¹¡£ IP ¥¢¥É¥ì¥¹¤ò»ý¤Á¤Þ¤¹¤¬¤Þ¤À»ÈÍÑ¤·¤Æ¤¤Þ¤»¤ó¡£ ¼¡¤ËÁ÷¿®¤µ¤ì¤ë¥á¥Ã¥»¡¼¥¸¤Ï DHCPREQUEST ¤Ç¤¢¤ê¡¢ ¤³¤ì¤Ï¥Ö¥í¡¼¥É¥¥ã¥¹¥È¤µ¤ì¤Þ¤¹¡£ ±þÅú¤¬²¿¤âÊ¹¤³¤¨Ê¤¤¤È¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ï¤³Î¥¢¥É¥ì¥¹¤Ë¥Ð¥¤¥ó¥É¤·¡¢ BOUND ¾õÂÖ¤ËÁ«°Ü¤·¤Þ¤¹¡£
- Select - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï SELECTING ¾õÂÖ¤Ç¤¹¡£ ¾¯¤Ê¤¯¤È¤â 1 ¸Ä¤Î DHCPOFFER ¥á¥Ã¥»¡¼¥¸¤Ï¼õ¿®¤·¤Þ¤·¤¿¤¬¡¢ Â¾¤Î DHCPOFFER ¥á¥Ã¥»¡¼¥¸¤òÂ¾¤Î¥µ¡¼¥Ð¤«¤é¼õ¤±¼è¤ë¤«¤É¤¦¤«¤ÂÔ¤Ã¤Æ¤¤¤Þ¤¹¡£ SELECTING ¾õÂÖ¤Ç¤Ï¥á¥Ã¥»¡¼¥¸¤ÏÁ÷¿®¤µ¤ì¤Þ¤»¤ó¡£
- Request - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï REQUESTING ¾õÂÖ¤Ç¤¹¡£ ¾¯¤Ê¤¯¤È¤â 1 ¸Ä¤Î DHCPOFFER ¥á¥Ã¥»¡¼¥¸¤ò¼õ¿®¤·¡¢ ¤½¤Î¤¦¤Á¤Î¤É¤ì¤«¤ò¥í×µá¤ë¤«Á²Âò¤·¤Þ¤·¤¿¡£ ¼¡¤ËÁ÷¿®¤µ¤ì¤ë¥á¥Ã¥»¡¼¥¸¤Ï DHCPREQUEST ¥á¥Ã¥»¡¼¥¸¤Ç¤¢¤ê¡¢ ¤³¤ì¤Ï¥Ö¥í¡¼¥É¥¥ã¥¹¥È¤µ¤ì¤Þ¤¹¡£
- Bound - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï BOUND ¾õÂÖ¤Ç¤¹¡£ IP ¥¢¥É¥ì¥¹¤ò½êÍ¤·¤Æ¤¤¤Þ¤¹¡£ ¤³¤Î¾õÂÖ¤Ç¤Ï¥á¥Ã¥»¡¼¥¸¤ÏÁ÷¿®¤µ¤ì¤Þ¤»¤ó¡£
- Renew - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï RENEWING ¾õÂÖ¤Ç¤¹¡£ IP ¥¢¥É¥ì¥¹¤ò½êÍ-¤·¤Æ¤ª¤ê¡¢¤³¤ì¤ò¹¿·¤¹¤ë¿¤á¤Ë¥µ¡¼¥Ð¤ËÀÜÂ³¤ò»îß¤Æ¤¤¤Þ¤¹¡£ ¼¡¤ËÁ÷¿®¤µ¤ì¤ë¥á¥Ã¥»¡¼¥¸¤Ï DHCPREQUEST ¥á¥Ã¥»¡¼¥¸¤Ç¤¢¤ê¡¢ ¤³¤ì¤Ï¥µ¡¼¥Ð¤ËÄ¾ÀÜ¥æ¥Ë¥¥ã¥¹¥È¤µ¤ì¤Þ¤¹¡£
- Rebind - DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï REBINDING ¾õÂÖ¤Ç¤¹¡£ IP ¥¢¥É¥ì¥¹¤ò½êÍ¤·¤Æ¤ª¤ê¡¢ ¤³¤ì¤ò¹¿·¤¹¤ë¿¤á¤ËÇ¤°Õ¤Î¥µ¡¼¥Ð¤ËÀÜÂ³¤ò»îß¤Æ¤¤¤Þ¤¹¡£ ¼¡¤ËÁ÷¿®¤µ¤ì¤ë¥á¥Ã¥»¡¼¥¸¤Ï DHCPREQUEST ¥á¥Ã¥»¡¼¥¸¤Ç¤¢¤ê¡¢ ¤³¤ì¤Ï¥Ö¥í¡¼¥É¥¥ã¥¹¥È¤µ¤ì¤Þ¤¹¡£

»²¾È: ¥í¥°

¥í¥°Ê¸¤ò»ÈÍÑ¤·¤Æ¡¢É¸½à¥í¥°¥Á¥ã¥Í¥ë¤Ë¾ðÊó¤òÁ÷¿®²ÄÇ½¤Ç¤¹¡£ ¥í¥°Ê¸¤Ï¡¢¾ÊÎ¬²ÄÇ½¤Ê priority (**fatal**, **error**, **info**, **debug** ¤Î¤¤ºº¤ì¤«) ¤È¡¢ ¥Ç¡¼¥¿¤ò¼è¤ê¤Þ¤¹¡£

log (*priority***,** *data-expr***)**

¥í¥°Ê¸¤Ï¡¢Ã±°ì¤Î¥Ç¡¼¥¿¿¿°°ú¿ô¤Î¤ß¼è¤ê¤Þ¤¹¡£ Ê£¿ô¤Î¥Ç¡¼¥¿¿ÃÍ¤ò½ÐÎÏ¤·¤¿¤¤¾ì¹ç¡¢ **concat** ¥ª¥Ú¥ì¡¼¥¿¤ò»ÈÍÑ¤·¤Æ¤½¤ì¤é¤òÏ¢·ë¤¤ëÉ¬Í×¤¬¤¢¤ê¤Þ¤¹¡£

»²¾È: Æ°Åª¤Ê DNS ¹¹¿·

DHCP ¥¯¥é¥¤¥¢¥ó¥È¤È¥µ¡¼¥Ð¤Ï¡¢ Æ°Åª¤Ë¥É¥á¥¤¥ó¥Í¡¼¥à¥·¥¹¥Æ¥à¤ò¹¹¿·¤¹¤ëÇ½ÎÏ¤¬¤¢¤ê¤Þ¤¹¡£ ÀßÄê¥Õ¥¡¥¤¥ëÃæ¤Ë¡¢¤É¤Î¤è¦¤Ë¥É¥á¥¤¥ó¥Í¡¼¥à¥·¥¹¥Æ¥à¤ò¹¹¿·¤·¤·¤Æ¡ß¤·¤¤¤«¡¢ ÄêµÁ²ÄÇ½¤Ç¤¹¡£ ¹¹¿·¤Ï RFC 2136 ¤Ë½¾¤Ã¤Æ¤¤¤ë¤¿¤á¡¢ RFC 2136 ¤ò¥µ¥Ý¡¼¥È¤¹¤ë DNS ¥µ¡¼¥Ð¤Ï¡¢ DHCP ¥µ¡¼¥Ð¤«¤é¤Î¹¹¿·¤ò¼õ¤±¤ÉÕ¤±²ÄÇ½È»×¤ï¤ì¤Þ¤¹¡£

¥»¥¥å¥ê¥Æ¥£

TSIG ¤ª¤è¤Ó DNSSEC ¤Ï¤Þ¤À¥µ¥Ý¡¼¥È¤µ¤ì¤Æ¤¤¤Þ¤»¤ó¡£ DHCP ¥µ¡¼¥Ð¤Þ¤¿¤Ï¥¯¥é¥¤¥¢¥ó¥È¤«¤é¤Î¹¹¿·¤ò¼õ¤±¤ÉÕ¤±¤ë¤è¦¤Ë DNS ¥µ¡¼¥Ð¤ò¤ÀßÄê¤¤¤ë¾ì¹ç¡¢¸¡¢Â¤ÎÌµ¤¤¹¹¿·¤ËÂÐ¤·¤Æ DNS ¥µ¡¼¥Ð¤ò»¯¤¹³¤È¤Ë¤Ê¤ë¤«¤â¤·¤ì¤Þ¤»¤ó¡£ ¤³¤ì¤òÈò±¤ë¤¿¤á¤Ëº£¤¹¤°¤Ç¤¤ëºÇÎÉ¤ÎÊýË¡¤Ï¡¢ IP ¥¢¥É¥ì¥¹¥Ù¡¼¥Î¥Ñ¥±¥Ã¥È¥Õ¥£¥ë¥¿¤ò»ÈÍÑ¤·¤Æ¡¢ ¸¢¸Â¤ÎÌµ¤¤¥Û¹¼¥È¤«¤é¤Î¹¹¿·¤Í×µáÈ¯¹Ô¤òÍÞ»ß¤¹¤ë¤³¤È¤Ç¤¹¡£ ÌÀ¤é¤«¤Ë¡¢½¾¤õ¤Ç¤Ï¥¯¥é¥¤¥¢¥ó¥È¤Î¹¹¿·¤ËÂÐ¤¹¤ë¥»¥¥å¥ê¥Æ¥£¤òÄó¶¡¤¹¤ëÊýË¡¤Ï ¤¢¤ê¤Þ¤»¤ó¡£ ¤³¤Î¿¤á¤Ë¤Ï TSIG ¤« DNSSEC ¤¬É¬Í×¤Ç¤¹¤¬¡¢ ¤³¤Î DHCP ÇÛÉÛÊª¤Ë¤Ï¤Þ¤À´Þ¤Þ¤ì¤Æ¤¤¤Þ¤»¤ó¡£

Æ°Åª DNS (DDNS) ¹¹¿·¤Ï¡¢**dns-update** ¼°¤ò»ÈÍÑ¤¹¤ë¤³¤È¤Ç¼Â¹Ô¤µ¤ì¤Þ¤¹¡£ **dns-update** ¼°¤Ï¡¢¥Ö¡¼¥ë¼°¤Ç¤¢¤ê¡¢4 ¸Ä¤Î¥Ñ¥é¥á¡¼¥¿¤ò¼è¤ê¤Þ¤¹¡£ ¹¹¿·¤È¥Ä®¸ù¹¤ë¤È¡¢¤ë²Ì¤Ï¿¿¤Ë¤Ê¤ë¤È¡¢¤ë²Ì¤Ïµ¶¤Ë¤Ê¤ë¤Þ¤¹¡£ 4 ¸Ä¤Î¥Ñ¥é¥á¡¼¥¿¤ê¥½¼¥¹¥ì¥³¡¼¥É¥¿¥¤¥× (RR)¡¢ RR ¤Î¸ÊÕ¡¢RR ¤Î±¦ÊÕ¡¢¥ì¥³¡¼¥É¤ËÅ¬ÍÑ¤µ¤ì¤ëÙ¤ ttl ¤Ç¤¹¡£ ¤³¤Î´Ø¿ô¤Î°Ç¤â´ÊÃ±¤Ê»ÈÍÑÎã¤Ï¡¢dhcpd.conf ¥Õ¥¡¥¤¥ë¤Î»²¾ÈÀá¤Ë¤¢¤ê¡¢ ¤Ê¤Ë¤¬¯¨¤¤ë¤«µ-½Ò¤µ¤ì¤Æ¤¤¤Þ¤¹¡£ ¤³¤ÎÎã¤Ç¤Ï¡¢Ê£¿ô¤Î¼°¤¬»ÈÍÑ¤µ¤ì¤Æ¡¢ **dns-update** ÍÑ¤Î°ú¿ô¤¬ºîÀ®¤µ¤ì¤Æ¤¤¤Þ¤¹¡£

Îã¤ÎÃæ¤Ç¤Ï¡¢ºÇ½é¤Î **dns-update** ¼°¤Ø¤Î 1 ÈÖÌÜ¤Î°ú¿ô¤Ï¡¢ A RR ¥¿¥¤¥×¤ËÉ¾²Á¤µ¤ì¤ë¥Ç¡¼¥¿¿¼°¤Ç¤¹¡£ 2 ÈÖÌÜ¤Î°ú¿ô¤Ï¡¢DHCP host-name ¥ª¥×¥·¥ç¥ó¤È ¥í¡¼¥«¥ë¥É¥á¥¤¥ó¡¢¤³¤Î¾ì¹ç "ssd.example.net"¡¢ ¤ò´Þ¤à¥Æ¥-¥¹¥È¥¹¥È¥ê¥ó¥°¤òÏ¢·ë¤¤ë¤³¤È¤Ç¡¢¹½ÃÛ¤µ¤ì¤Þ¤¹¡£ 3 ÈÖÌÜ¤Î°ú¿ô¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È³ä¤êÂö¤Æ¤é¤ì¿¢¥É¥ì¥¹¤ò¡¢ 32 ¥Ó¥Ã¥È¤Î¿ôÃí¤«¤é³Æ¥Ð¥¤¥È¤¤ "." ¤Ç¶èÀÚ¤Ã¤¿ ASCII Ê¸»úÎó¤ËÊÑ´¹¤¤ë¤³¤È¤Ç¡¢ ¹½ÃÛ¤µ¤ì¤Þ¤¹¡£ 4 ÈÖÌÜ¤Î°ú¿ô TTL ¤Ï¡¢¥ê¼¥¹¤Î»Ä¤ê»þ´Ö¤Ç¤¹ (¤³¤ì¤ÏËÜÅö¤ÏÀµ¤·¤¯¤¢¤ê¤Þ¤»¤ó¡£ ¤Ê¤¼¤Ê¤é DNS ¥µ¡¼¥Ð¤Ï¡¢Í×µá¤ËÂÐ¤·¤Æ¤¤Ä¤â³Î TTL ÃÍ¤ò½ÐÎÏ¤·¤Æ¤·¤Þ¤¦«¤é¤Ç¤¹¡£ ¤³¤ì¤Ï¡¢¥ê¼¥´ü¸ÂÀÚ¤ì¤Î¿ôÉÃÁ°¤Ç¤¢¤Ã¤Æ¤â¤Ç¤¹)¡£

ºÇ½é¤Î **dns-update** Ê¸¤¬À®¸ù¤¤ë¤È¡¢ °ú¤Â³¤¤¤Æ 2 ÈÖÌÜ¤Î¹¹¿·¤Ë¤è¤ê PTR RR ¤¬¥¤¥ó¥¹¥È¡¼¥ë¤µ¤ì¤Þ¤¹¡£ PTR ¥ì¥³¡¼¥É¤Î¥¤¥ó¥¹¥È¡¼¥ë¤Ï¡¢A RR ¤Î¥¤¥ó¥¹¥È¡¼¥ë¤ÈÆ±¸ÍÍ¤Ç¤¤¤¬¡¢ ¥ì¥³¡¼¥É¤Îº¸ÊÕ¤Ï¥ê¼¥µ¤¤¿¢¥É¥ì¥¹¤òµÕ¤Ë¤·¤Æ ".in-addr.arpa" ¤È ·ë¹ç¤µ¤ì¤¿¤ã¤Î¤Ç¤¹¡£ ±¦ÊÕ¤Ï¡¢¥¢¥É¥ì¥¹¤Î¥ê¼¥Äó¶¡¤Àè¥¯¥é¥¢¥ó¥Î¡¢´°Á´¤Ê·Á¤Ç¤Î¥É¥á¥¤¥óÌ¾¤Ç¤¹¡£

´ØÏ¢¹àÌÜ

dhcpd.conf(5), dhcpd.leases(5), dhclient.conf(5), dhcp-eval(5), dhcpd(8), dhclient(8), RFC2132, RFC2131

ºî¼Ô

Internet Systems Consortium DHCP Distribution ¤Ï¡¢Vixie Labs ¤È¤Î·ÀÌó¤Î¤â¤È¤Ç¡¢Ted Lemon ¤¬µ-½Ò¤·¤Þ¤·¤¿¡£ ËÜ¥×¥í¥¸¥§¯¥È¤Î»ñ¶â¤Ï¡¢Internet Systems Consortium ¤¬Äó¶¡¤·¤Þ¤·¤¿¡£ Internet Systems Consortium ¤Ë´Ø¤¹¤ë¾ðÊó¤Ï¡¢ **http://www.isc.org** ¤Ë¤¢¤ê¤Þ¤¹¡£

NAME

dhcp-options - Dynamic Host Configuration Protocol options

DESCRIPTION

The Dynamic Host Configuration protocol allows the client to receive **options** from the DHCP server describing the network configuration and various services that are available on the network. When configuring **dhcpd(8)** or **dhclient(8) ,** options must often be declared. The syntax for declaring options, and the names and formats of the options that can be declared, are documented here.

REFERENCE: OPTION STATEMENTS

DHCP *option* statements always start with the *option* keyword, followed by an option name, followed by option data. The option names and data formats are described below. It is not necessary to exhaustively specify all DHCP options - only those options which are needed by clients must be specified.

Option data comes in a variety of formats, as defined below:

The **ip-address** data type can be entered either as an explicit IP address (e.g., 239.254.197.10) or as a domain name (e.g., haagen.isc.org). When entering a domain name, be sure that that domain name resolves to a single IP address.

The **int32** data type specifies a signed 32-bit integer. The **uint32** data type specifies an unsigned 32-bit integer. The **int16** and **uint16** data types specify signed and unsigned 16-bit integers. The **int8** and **uint8** data types specify signed and unsigned 8-bit integers. Unsigned 8-bit integers are also sometimes referred to as octets.

The **text** data type specifies an NVT ASCII string, which must be enclosed in double quotes - for example, to specify a root-path option, the syntax would be

```
option root-path "10.0.1.4:/var/tmp/rootfs";
```

The **domain-name** data type specifies a domain name, which must not enclosed in double quotes. This data type is not used for any existing DHCP options. The domain name is stored just as if it were a text option.

The **flag** data type specifies a boolean value. Booleans can be either true or false (or on or off, if that makes more sense to you).

The **string** data type specifies either an NVT ASCII string enclosed in double quotes, or a series of octets specified in hexadecimal, separated by colons. For example:

```
 option dhcp-client-identifier "CLIENT-FOO";
or
 option dhcp-client-identifier 43:4c:49:45:54:2d:46:4f:4f;
```

SETTING OPTION VALUES USING EXPRESSIONS

Sometimes it's helpful to be able to set the value of a DHCP option based on some value that the client has sent. To do this, you can use expression evaluation. The **dhcp-eval(5)** manual page describes how to write expressions. To assign the result of an evaluation to an option, define the option as follows:

option *my-option* **=** *expression* **;**

For example:

```
option hostname = binary-to-ascii (16, 8, "-",
                                   substring (hardware, 1, 6));
```

STANDARD DHCP OPTIONS

The documentation for the various options mentioned below is taken from the latest IETF draft document on DHCP options. Options not listed below may not yet be implemented, but it is possible to use such options by defining them in the configuration file. Please see the DEFINING NEW OPTIONS heading later in this document for more information.

Some of the options documented here are automatically generated by the DHCP server or by clients, and cannot be configured by the user. The value of such an option can be used in the configuration file of the receiving DHCP protocol agent (server or client), for example in conditional expressions. However, the value of the option cannot be used in the configuration file of the sending agent, because the value is determined only *after* the configuration file has been processed. In the following documentation, such options will be shown as "not user configurable"

The standard options are:

option all-subnets-local *flag***;**

This option specifies whether or not the client may assume that all subnets of the IP network to which the client is connected use the same MTU as the subnet of that network to which the client is directly connected. A value of true indicates that all subnets share the same MTU. A value of false means that the client should assume that some subnets of the directly connected network may have smaller MTUs.

option arp-cache-timeout *uint32***;**

This option specifies the timeout in seconds for ARP cache entries.

option bootfile-name *text***;**

This option is used to identify a bootstrap file. If supported by the client, it should have the same effect as the **filename** declaration. BOOTP clients are unlikely to support this option. Some DHCP clients will support it, and others actually require it.

option boot-size *uint16***;**

This option specifies the length in 512-octet blocks of the default boot image for the client.

option broadcast-address *ip-address***;**

This option specifies the broadcast address in use on the client's subnet. Legal values for broadcast addresses are specified in section 3.2.1.3 of STD 3 (RFC1122).

option cookie-servers *ip-address* **[***, ip-address...* **];**

The cookie server option specifies a list of RFC 865 cookie servers available to the client. Servers should be listed in order of preference.

option default-ip-ttl *uint8;*

This option specifies the default time-to-live that the client should use on outgoing datagrams.

option default-tcp-ttl *uint8***;**

This option specifies the default TTL that the client should use when sending TCP segments. The minimum value is 1.

option dhcp-client-identifier *string***;**

This option can be used to specify a DHCP client identifier in a host declaration, so that dhcpd can find the host record by matching against the client identifier.

Please be aware that some DHCP clients, when configured with client identifiers that are ASCII text, will prepend a zero to the ASCII text. So you may need to write:

```
option dhcp-client-identifier "\0foo";
```

rather than:

```
option dhcp-client-identifier "foo";
```

option dhcp-lease-time *uint32***;**

This option is used in a client request (DHCPDISCOVER or DHCPREQUEST) to allow the client to request a lease time for the IP address. In a server reply (DHCPOFFER), a DHCP server uses this option to specify the lease time it is willing to offer.

This option is not directly user configurable in the server; refer to the *max-lease-time* and *default-lease-time* server options in **dhcpd.conf(5).**

option dhcp-max-message-size *uint16***;**

This option, when sent by the client, specifies the maximum size of any response that the server sends to the client. When specified on the server, if the client did not send a dhcp-max-message-size option, the size specified on the server is used. This works for BOOTP as well as DHCP responses.

option dhcp-message *text***;**

This option is used by a DHCP server to provide an error message to a DHCP client in a DHCPNAK message in the event of a failure. A client may use this option in a DHCPDECLINE message to indicate why the client declined the offered parameters.

This option is not user configurable.

option dhcp-message-type *uint8***;**

This option, sent by both client and server, specifies the type of DHCP message contained in the DHCP packet. Possible values (taken directly from RFC2132) are:

1 DHCPDISCOVER
2 DHCPOFFER
3 DHCPREQUEST
4 DHCPDECLINE
5 DHCPACK
6 DHCPNAK
7 DHCPRELEASE
8 DHCPINFORM

This option is not user configurable.

option dhcp-option-overload *uint8***;**

This option is used to indicate that the DHCP 'sname' or 'file' fields are being overloaded by using them to carry DHCP options. A DHCP server inserts this option if the returned parameters will exceed the usual space allotted for options.

If this option is present, the client interprets the specified additional fields after it concludes interpretation of the standard option fields.

Legal values for this option are:

1 the 'file' field is used to hold options
2 the 'sname' field is used to hold options
3 both fields are used to hold options

This option is not user configurable.

option dhcp-parameter-request-list *uint16***;**

This option, when sent by the client, specifies which options the client wishes the server to return. Normally, in the ISC DHCP client, this is done using the *request* statement. If this option is not specified by the client, the DHCP server will normally return every option that is valid in scope and that fits into the reply. When this option is specified on the server, the server returns the specified options. This can be used to force a client to take options that it hasn't requested, and it can also be used to tailor the response of the DHCP server for clients that may need a more limited set of options than those the server would normally return.

option dhcp-rebinding-time *uint32***;**

This option specifies the number of seconds from the time a client gets an address until the client transitions to the REBINDING state.

This option is not user configurable.

option dhcp-renewal-time *uint32***;**

This option specifies the number of seconds from the time a client gets an address until the client transitions to the RENEWING state.

This option is not user configurable.

option dhcp-requested-address *ip-address***;**

This option is used by the client in a DHCPDISCOVER to request that a particular IP address be assigned.

This option is not user configurable.

option dhcp-server-identifier *ip-address***;**

This option is used in DHCPOFFER and DHCPREQUEST messages, and may optionally be included in the DHCPACK and DHCPNAK messages. DHCP servers include this option in the DHCPOFFER in order to allow the client to distinguish between lease offers. DHCP clients use the contents of the 'server identifier' field as the destination address for any DHCP messages unicast to the DHCP server. DHCP clients also indicate which of several lease offers is being accepted by including this option in a DHCPREQUEST message.

The value of this option is the IP address of the server.

This option is not directly user configurable. See the *server-identifier* server option in *dhcpd.conf(5).*

option domain-name *text***;**

This option specifies the domain name that client should use when resolving hostnames via the Domain Name System.

option domain-name-servers *ip-address* [**,** *ip-address...*]**;**

The domain-name-servers option specifies a list of Domain Name System (STD 13, RFC 1035) name servers available to the client. Servers should be listed in order of preference.

option extensions-path *text***;**

This option specifies the name of a file containing additional options to be interpreted according to the DHCP option format as specified in RFC2132.

option finger-server *ip-address* [**,** *ip-address...*]**;**

The Finger server option specifies a list of Finger servers available to the client. Servers should be listed in order of preference.

option font-servers *ip-address* [**,** *ip-address...*]**;**

This option specifies a list of X Window System Font servers available to the client. Servers should be listed in order of preference.

option host-name *string***;**

This option specifies the name of the client. The name may or may not be qualified with the local domain name (it is preferable to use the domain-name option to specify the domain name). See RFC 1035 for character set restrictions. This option is only honored by **dhclient-script(8)** if the hostname for the client machine is not set.

option ieee802-3-encapsulation *flag***;**

This option specifies whether or not the client should use Ethernet Version 2 (RFC 894) or IEEE 802.3 (RFC 1042) encapsulation if the interface is an Ethernet. A value of false indicates that the client should use RFC 894 encapsulation. A value of true means that the client should use RFC 1042 encapsulation.

option ien116-name-servers *ip-address* [**,** *ip-address...*];

The ien116-name-servers option specifies a list of IEN 116 name servers available to the client. Servers should be listed in order of preference.

option impress-servers *ip-address* [**,** *ip-address*...]**;**

The impress-server option specifies a list of Imagen Impress servers available to the client. Servers should be listed in order of preference.

option interface-mtu *uint16***;**

This option specifies the MTU to use on this interface. The minimum legal value for the MTU is 68.

option ip-forwarding *flag***;**

This option specifies whether the client should configure its IP layer for packet forwarding. A value of false means disable IP forwarding, and a value of true means enable IP forwarding.

option irc-server *ip-address* [**,** *ip-address*...]**;**

The IRC server option specifies a list of IRC servers available to the client. Servers should be listed in order of preference.

option log-servers *ip-address* [**,** *ip-address*...]**;**

The log-server option specifies a list of MIT-LCS UDP log servers available to the client. Servers should be listed in order of preference.

option lpr-servers *ip-address* [**,** *ip-address*...]**;**

The LPR server option specifies a list of RFC 1179 line printer servers available to the client. Servers should be listed in order of preference.

option mask-supplier *flag***;**

This option specifies whether or not the client should respond to subnet mask requests using ICMP. A value of false indicates that the client should not respond. A value of true means that the client should respond.

option max-dgram-reassembly *uint16***;**

This option specifies the maximum size datagram that the client should be prepared to reassemble. The minimum legal value is 576.

option merit-dump *text***;**

This option specifies the path-name of a file to which the client's core image should be dumped in the event the client crashes. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

option mobile-ip-home-agent *ip-address* [**,** *ip-address*...]**;**

This option specifies a list of IP addresses indicating mobile IP home agents available to the client. Agents should be listed in order of preference, although normally there will be only one such agent.

option nds-context *string***;**

The nds-context option specifies the name of the initial Netware Directory Service for an NDS client.

option nds-servers *ip-address* [**,** *ip-address*...]**;**

The nds-servers option specifies a list of IP addresses of NDS servers.

option nds-tree-name *string***;**

The nds-tree-name option specifies NDS tree name that the NDS client should use.

option netbios-dd-server *ip-address* [**,** *ip-address*...]**;**

The NetBIOS datagram distribution server (NBDD) option specifies a list of RFC 1001/1002 NBDD servers listed in order of preference.

option netbios-name-servers *ip-address* [**,** *ip-address*...]**;**

The NetBIOS name server (NBNS) option specifies a list of RFC 1001/1002 NBNS name servers listed in order of preference. NetBIOS Name Service is currently more commonly referred to as WINS. WINS servers can be specified using the netbios-name-servers option.

option netbios-node-type *uint8***;**

The NetBIOS node type option allows NetBIOS over TCP/IP clients which are configurable to be configured as described in RFC 1001/1002. The value is specified as a single octet which identifies the client type.

Possible node types are:

1 B-node: Broadcast - no WINS

2 P-node: Peer - WINS only

4 M-node: Mixed - broadcast, then WINS

8 H-node: Hybrid - WINS, then broadcast

option netbios-scope *string***;**

The NetBIOS scope option specifies the NetBIOS over TCP/IP scope parameter for the client as specified in RFC 1001/1002. See RFC1001, RFC1002, and RFC1035 for character-set restrictions.

option nis-domain *text***;**

This option specifies the name of the client's NIS (Sun Network Information Services) domain. The domain is formatted as a character string consisting of characters from the NVT ASCII character set.

option nis-servers *ip-address* [**,** *ip-address...*]**;**

This option specifies a list of IP addresses indicating NIS servers available to the client. Servers should be listed in order of preference.

option nisplus-domain *text***;**

This option specifies the name of the client's NIS+ domain. The domain is formatted as a character string consisting of characters from the NVT ASCII character set.

option nisplus-servers *ip-address* [**,** *ip-address...*]**;**

This option specifies a list of IP addresses indicating NIS+ servers available to the client. Servers should be listed in order of preference.

option nntp-server *ip-address* [**,** *ip-address...*]**;**

The NNTP server option specifies a list of NNTP servesr available to the client. Servers should be listed in order of preference.

option non-local-source-routing *flag***;**

This option specifies whether the client should configure its IP layer to allow forwarding of datagrams with non-local source routes (see Section 3.3.5 of [4] for a discussion of this topic). A value of false means disallow forwarding of such datagrams, and a value of true means allow forwarding.

option ntp-servers *ip-address* [**,** *ip-address...*]**;**

This option specifies a list of IP addresses indicating NTP (RFC 1035) servers available to the client. Servers should be listed in order of preference.

option nwip-domain *string***;**

The name of the NetWare/IP domain that a NetWare/IP client should use.

option nwip-suboptions *string***;**

A sequence of suboptions for NetWare/IP clients - see RFC2242 for details. Normally this option is set by specifying specific NetWare/IP suboptions - see the NETWARE/IP SUBOPTIONS section for more information.

option path-mtu-aging-timeout *uint32***;**

This option specifies the timeout (in seconds) to use when aging Path MTU values discovered by the mechanism defined in RFC 1191.

option path-mtu-plateau-table *uint16* [**,** *uint16...*]**;**

This option specifies a table of MTU sizes to use when performing Path MTU Discovery as defined in RFC 1191. The table is formatted as a list of 16-bit unsigned integers, ordered from smallest to largest. The minimum MTU value cannot be smaller than 68.

option perform-mask-discovery *flag***;**

This option specifies whether or not the client should perform subnet mask discovery using ICMP. A value of false indicates that the client should not perform mask discovery. A value of true means that the client should perform mask discovery.

option policy-filter *ip-address ip-address*
[**,** *ip-address ip-address...*]**;**

This option specifies policy filters for non-local source routing. The filters consist of a list of IP addresses and masks which specify destination/mask pairs with which to filter incoming source routes.

Any source routed datagram whose next-hop address does not match one of the filters should be discarded by the client.

See STD 3 (RFC1122) for further information.

option pop-server *ip-address* [**,** *ip-address...*]**;**

The POP3 server option specifies a list of POP3 servers available to the client. Servers should be listed in order of preference.

option resource-location-servers *ip-address*
[**,** *ip-address...*]**;**

This option specifies a list of RFC 887 Resource Location servers available to the client. Servers should be listed in order of preference.

option root-path *text***;**

This option specifies the path-name that contains the client's root disk. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

option router-discovery *flag***;**

This option specifies whether or not the client should solicit routers using the Router Discovery mechanism defined in RFC 1256. A value of false indicates that the client should not perform router discovery. A value of true means that the client should perform router discovery.

option router-solicitation-address *ip-address***;**

This option specifies the address to which the client should transmit router solicitation requests.

option routers *ip-address* [**,** *ip-address...*]**;**

The routers option specifies a list of IP addresses for routers on the client's subnet. Routers should be listed in order of preference.

option slp-directory-agent *boolean ip-address* [**,** *ip-address...*]**;**

This option specifies two things: the IP addresses of one or more Service Location Protocol Directory Agents, and whether the use of these addresses is mandatory. If the initial boolean value is true, the SLP agent should just use the IP addresses given. If the value is false, the SLP agent may additionally do active or passive multicast discovery of SLP agents (see RFC2165 for details).

Please note that in this option and the slp-service-scope option, the term "SLP Agent" is being used to refer to a Service Location Protocol agent running on a machine that is being configured using the DHCP protocol.

Also, please be aware that some companies may refer to SLP as NDS. If you have an NDS directory agent whose address you need to configure, the slp-directory-agent option should work.

option slp-service-scope *boolean text***;**

The Service Location Protocol Service Scope Option specifies two things: a list of service scopes for SLP, and whether the use of this list is mandatory. If the initial boolean value is true, the SLP agent should only use the list of scopes provided in this option; otherwise, it may use its own static configuration in preference to the list provided in this option.

The text string should be a comma-separated list of scopes that the SLP agent should use. It may be omitted, in which case the SLP Agent will use the aggregated list of scopes of all directory agents known to the SLP agent.

option smtp-server *ip-address* [**,** *ip-address...*]**;**

The SMTP server option specifies a list of SMTP servers available to the client. Servers should be listed in order of preference.

option static-routes *ip-address ip-address*
[**,** *ip-address ip-address...*]**;**

This option specifies a list of static routes that the client should install in its routing cache. If multiple routes to the same destination are specified, they are listed in descending order of priority.

The routes consist of a list of IP address pairs. The first address is the destination address, and the second address is the router for the destination.

The default route (0.0.0.0) is an illegal destination for a static route. To specify the default route, use the **routers** option. Also, please note that this option is not intended for classless IP routing - it does not include a subnet mask. Since classless IP routing is now the most widely deployed routing standard, this option is virtually useless, and is not implemented by any of the popular DHCP clients, for example the Microsoft DHCP client.

option streettalk-directory-assistance-server *ip-address*
[**,** *ip-address...*]**;**

The StreetTalk Directory Assistance (STDA) server option specifies a list of STDA servers available to the client. Servers should be listed in order of preference.

option streettalk-server *ip-address* [**,** *ip-address...*]**;**

The StreetTalk server option specifies a list of StreetTalk servers available to the client. Servers should be listed in order of preference.

option subnet-mask *ip-address***;**

The subnet mask option specifies the client's subnet mask as per RFC 950. If no subnet mask option is provided anywhere in scope, as a last resort dhcpd will use the subnet mask from the subnet declaration for the network on which an address is being assigned. However, *any* subnet-mask option declaration that is in scope for the address being assigned will override the subnet mask specified in the subnet declaration.

option subnet-selection *string***;**

Sent by the client if an address is required in a subnet other than the one that would normally be selected (based on the relaying address of the connected subnet the request is obtained from). See RFC3011. Note that the option number used by this server is 118; this has not always been the defined number, and some clients may use a different value. Use of this option should be regarded as slightly experimental!

This option is not user configurable in the server.

option swap-server *ip-address***;**

This specifies the IP address of the client's swap server.

option tcp-keepalive-garbage *flag***;**

This option specifies whether or not the client should send TCP keepalive messages with an octet of garbage for compatibility with older implementations. A value of false indicates that a garbage octet should not be sent. A value of true indicates that a garbage octet should be sent.

option tcp-keepalive-interval *uint32***;**

This option specifies the interval (in seconds) that the client TCP should wait before sending a keepalive message on a TCP connection. The time is specified as a 32-bit unsigned integer. A value of zero indicates that the client should not generate keepalive messages on connections unless specifically requested by an application.

option tftp-server-name *text***;**

This option is used to identify a TFTP server and, if supported by the client, should have the same effect as the **server-name** declaration. BOOTP clients are unlikely to support this option. Some DHCP clients will support it, and others actually require it.

option time-offset *int32***;**

The time-offset option specifies the offset of the client's subnet in seconds from Coordinated Universal Time (UTC).

option time-servers *ip-address* [, *ip-address*...]**;**

The time-server option specifies a list of RFC 868 time servers available to the client. Servers should be listed in order of preference.

option trailer-encapsulation *flag***;**

This option specifies whether or not the client should negotiate the use of trailers (RFC 893 [14]) when using the ARP protocol. A value of false indicates that the client should not attempt to use trailers. A value of true means that the client should attempt to use trailers.

option uap-servers *text***;**

This option specifies a list of URLs, each pointing to a user authentication service that is capable of processing authentication requests encapsulated in the User Authentication Protocol (UAP). UAP servers can accept either HTTP 1.1 or SSLv3 connections. If the list includes a URL that does not contain a port component, the normal default port is assumed (i.e., port 80 for http and port 443 for https). If the list includes a URL that does not contain a path component, the path /uap is assumed. If more than one URL is specified in this list, the URLs are separated by spaces.

option user-class *string***;**

This option is used by some DHCP clients as a way for users to specify identifying information to the client. This can be used in a similar way to the vendor-class-identifier option, but the value of the option is specified by the user, not the vendor. Most recent DHCP clients have a way in the user interface to specify the value for this identifier, usually as a text string.

option vendor-class-identifier *string***;**

This option is used by some DHCP clients to identify the vendor type and possibly the configuration of a DHCP client. The information is a string of bytes whose contents are specific to the vendor and are not specified in a standard. To see what vendor class identifier clients are sending, you can write the following in your DHCP server configuration file:

set vendor-string = option vendor-class-identifier;

This will result in all entries in the DHCP server lease database file for clients that sent vendor-class-identifier options having a set statement that looks something like this:

set vendor-string = "SUNW.Ultra-5_10";

The vendor-class-identifier option is normally used by the DHCP server to determine the options that are returned in the **vendor-encapsulated-options** option. Please see the VENDOR

ENCAPSULATED OPTIONS section later in this manual page for further information.

option vendor-encapsulated-options *string***;**

The **vendor-encapsulated-options** option can contain either a single vendor-specific value or one or more vendor-specific suboptions. This option is not normally specified in the DHCP server configuration file - instead, a vendor class is defined for each vendor, vendor class suboptions are defined, values for those suboptions are defined, and the DHCP server makes up a response on that basis.

Some default behaviours for well-known DHCP client vendors (currently, the Microsoft Windows 2000 DHCP client) are configured automatically, but otherwise this must be configured manually - see the VENDOR ENCAPSULATED OPTIONS section later in this manual page for details.

option www-server *ip-address* [**,** *ip-address*...]**;**

The WWW server option specifies a list of WWW servers available to the client. Servers should be listed in order of preference.

option x-display-manager *ip-address* [**,** *ip-address*...]**;**

This option specifies a list of systems that are running the X Window System Display Manager and are available to the client. Addresses should be listed in order of preference.

RELAY AGENT INFORMATION OPTION

An IETF draft, draft-ietf-dhc-agent-options-11.txt, defines a series of encapsulated options that a relay agent can add to a DHCP packet when relaying it to the DHCP server. The server can then make address allocation decisions (or whatever other decisions it wants) based on these options. The server also returns these options in any replies it sends through the relay agent, so that the relay agent can use the information in these options for delivery or accounting purposes.

The current draft defines two options. To reference these options in the dhcp server, specify the option space name, "agent", followed by a period, followed by the option name. It is not normally useful to define values for these options in the server, although it is permissible. These options are not supported in the client.

option agent.circuit-id *string***;**

The circuit-id suboption encodes an agent-local identifier of the circuit from which a DHCP client-to-server packet was received. It is intended for use by agents in relaying DHCP responses back to the proper circuit. The format of this option is currently defined to be vendor-dependent, and will probably remain that way, although the current draft allows for for the possibility of standardizing the format in the future.

option agent.remote-id *string***;**

The remote-id suboption encodes information about the remote host end of a circuit. Examples of what it might contain include caller ID information, username information, remote ATM address, cable modem ID, and similar things. In principal, the meaning is not well-specified, and it should generally be assumed to be an opaque object that is administratively guaranteed to be unique to a particular remote end of a circuit.

THE CLIENT FQDN SUBOPTIONS

The Client FQDN option, currently defined in the Internet Draft draft-ietf-dhc-fqdn-option-00.txt is not a standard yet, but is in sufficiently wide use already that we have implemented it. Due to the complexity of the option format, we have implemented it as a suboption space rather than a single option. In general this option should not be configured by the user - instead it should be used as part of an automatic DNS update system.

option fqdn.no-client-update *flag***;**

When the client sends this, if it is true, it means the client will not attempt to update its A record. When sent by the server to the client, it means that the client *should not* update its own A record.

option fqdn.server-update *flag***;**

When the client sends this to the server, it is requesting that the server update its A record. When sent by the server, it means that the server has updated (or is about to update) the client's A record.

option fqdn.encoded *flag***;**

If true, this indicates that the domain name included in the option is encoded in DNS wire format, rather than as plain ASCII text. The client normally sets this to false if it doesn't support DNS wire format in the FQDN option. The server should always send back the same value that the client sent. When this value is set on the configuration side, it controls the format in which the *fqdn.fqdn* suboption is encoded.

option fqdn.rcode1 *flag***;**

option fqdn.rcode2 *flag***;**

These options specify the result of the updates of the A and PTR records, respectively, and are only sent by the DHCP server to the DHCP client. The values of these fields are those defined in the DNS protocol specification.

option fqdn.fqdn *text***;**

Specifies the domain name that the client wishes to use. This can be a fully-qualified domain name, or a single label. If there is no trailing generally update that name in some locally-defined domain.

option fqdn.hostname *--never set--***;**

This option should never be set, but it can be read back using the **option** and **config-option** operators in an expression, in which case it returns the first label in the **fqdn.fqdn** suboption - for example, if the value of **fqdn.fqdn** is "foo.example.com.", then **fqdn.hostname** will be "foo".

option fqdn.domainname *--never set--***;**

This option should never be set, but it can be read back using the **option** and **config-option** operators in an expression, in which case it returns all labels after the first label in the **fqdn.fqdn** suboption - for example, if the value of **fqdn.fqdn** is "foo.example.com.", then **fqdn.hostname** will be "example.com.". If this suboption value is not set, it means that an unqualified name was sent in the fqdn option, or that no fqdn option was sent at all.

If you wish to use any of these suboptions, we strongly recommend that you refer to the Client FQDN option draft (or standard, when it becomes a standard) - the documentation here is sketchy and incomplete in comparison, and is just intended for reference by people who already understand the Client FQDN option specification.

THE NETWARE/IP SUBOPTIONS

RFC2242 defines a set of encapsulated options for Novell NetWare/IP clients. To use these options in the dhcp server, specify the option space name, "nwip", followed by a period, followed by the option name. The following options can be specified:

option nwip.nsq-broadcast *flag***;**

If true, the client should use the NetWare Nearest Server Query to locate a NetWare/IP server. The behaviour of the Novell client if this suboption is false, or is not present, is not specified.

option nwip.preferred-dss *ip-address* [**,** *ip-address...*]**;**

This suboption specifies a list of up to five IP addresses, each of which should be the IP address of a NetWare Domain SAP/RIP server (DSS).

option nwip.nearest-nwip-server *ip-address*
[**,** *ip-address...*]**;**

This suboption specifies a list of up to five IP addresses, each of which should be the IP address of a Nearest NetWare IP server.

option nwip.autoretries *uint8***;**

Specifies the number of times that a NetWare/IP client should attempt to communicate with a given

DSS server at startup.

option nwip.autoretry-secs *uint8***;**

Specifies the number of seconds that a Netware/IP client should wait between retries when attempting to establish communications with a DSS server at startup.

option nwip.nwip-1-1 *uint8***;**

If true, the NetWare/IP client should support NetWare/IP version 1.1 compatibility. This is only needed if the client will be contacting Netware/IP version 1.1 servers.

option nwip.primary-dss *ip-address***;**

Specifies the IP address of the Primary Domain SAP/RIP Service server (DSS) for this NetWare/IP domain. The NetWare/IP administration utility uses this value as Primary DSS server when configuring a secondary DSS server.

DEFINING NEW OPTIONS

The Internet Systems Consortium DHCP client and server provide the capability to define new options. Each DHCP option has a name, a code, and a structure. The name is used by you to refer to the option. The code is a number, used by the DHCP server and client to refer to an option. The structure describes what the contents of an option looks like.

To define a new option, you need to choose a name for it that is not in use for some other option - for example, you can't use "host-name" because the DHCP protocol already defines a host-name option, which is documented earlier in this manual page. If an option name doesn't appear in this manual page, you can use it, but it's probably a good idea to put some kind of unique string at the beginning so you can be sure that future options don't take your name. For example, you might define an option, "local-host-name", feeling some confidence that no official DHCP option name will ever start with "local".

Once you have chosen a name, you must choose a code. For site-local options, all codes between 128 and 254 are reserved for DHCP options, so you can pick any one of these. In practice, some vendors have interpreted the protocol rather loosely and have used option code values greater than 128 themselves. There's no real way to avoid this problem, but it's not likely to cause too much trouble in practice.

The structure of an option is simply the format in which the option data appears. The ISC DHCP server currently supports a few simple types, like integers, booleans, strings and IP addresses, and it also supports the ability to define arrays of single types or arrays of fixed sequences of types.

New options are declared as follows:

option *new-name* **code** *new-code* **=** *definition* **;**

The values of *new-name* and *new-code* should be the name you have chosen for the new option and the code you have chosen. The *definition* should be the definition of the structure of the option.

The following simple option type definitions are supported:

BOOLEAN

option *new-name* **code** *new-code* **= boolean ;**

An option of type boolean is a flag with a value of either on or off (or true or false). So an example use of the boolean type would be:

```
option use-zephyr code 180 = boolean;
option use-zephyr on;
```

INTEGER

option *new-name* **code** *new-code* **=** *sign* **integer** *width* **;**

The *sign* token should either be blank, *unsigned* or *signed*. The width can be either 8, 16 or 32, and refers to the number of bits in the integer. So for example, the following two lines show a definition of the sql-connection-max option and its use:

```
option sql-connection-max code 192 = unsigned integer 16;
option sql-connection-max 1536;
```

IP-ADDRESS

option *new-name* **code** *new-code* **= ip-address ;**

An option whose structure is an IP address can be expressed either as a domain name or as a dotted quad. So the following is an example use of the ip-address type:

```
option sql-server-address code 193 = ip-address;
option sql-server-address sql.example.com;
```

TEXT

option *new-name* **code** *new-code* **= text ;**

An option whose type is text will encode an ASCII text string. For example:

```
option sql-default-connection-name code 194 = text;
option sql-default-connection-name "PRODZA";
```

DATA STRING

option *new-name* **code** *new-code* **= string ;**

An option whose type is a data string is essentially just a collection of bytes, and can be specified either as quoted text, like the text type, or as a list of hexadecimal contents separated by colons whose values must be between 0 and FF. For example:

```
option sql-identification-token code 195 = string;
option sql-identification-token 17:23:19:a6:42:ea:99:7c:22;
```

ENCAPSULATION

option *new-name* **code** *new-code* **= encapsulate** *identifier* **;**

An option whose type is **encapsulate** will encapsulate the contents of the option space specified in *identifier*. Examples of encapsulated options in the DHCP protocol as it currently exists include the vendor-encapsulated-options option, the netware-suboptions option and the relay-agent-information option.

```
option space local;
option local.demo code 1 = text;
option local-encapsulation code 197 = encapsulate local;
option local.demo "demo";
```

ARRAYS

Options can contain arrays of any of the above types except for the text and data string types, which aren't currently supported in arrays. An example of an array definition is as follows:

```
option kerberos-servers code 200 = array of ip-address;
option kerberos-servers 10.20.10.1, 10.20.11.1;
```

RECORDS

Options can also contain data structures consisting of a sequence of data types, which is sometimes called a record type. For example:

```
option contrived-001 code 201 = { boolean, integer 32, text };
option contrived-001 on 1772 "contrivance";
```

It's also possible to have options that are arrays of records, for example:

```
option new-static-routes code 201 = array of {
        ip-address, ip-address, ip-address, integer 8 };
option static-routes
        10.0.0.0 255.255.255.0 net-0-rtr.example.com 1,
        10.0.1.0 255.255.255.0 net-1-rtr.example.com 1,
        10.2.0.0 255.255.224.0 net-2-0-rtr.example.com 3;
```

VENDOR ENCAPSULATED OPTIONS

The DHCP protocol defines the **vendor-encapsulated-options** option, which allows vendors to define their own options that will be sent encapsulated in a standard DHCP option. The format of the **vendor-encapsulated-options** option is either a series of bytes whose format is not specified, or a sequence of options, each of which consists of a single-byte vendor-specific option code, followed by a single-byte length, followed by as many bytes of data as are specified in the length (the length does not include itself or the option code).

The value of this option can be set in one of two ways. The first way is to simply specify the data directly, using a text string or a colon-separated list of hexadecimal values. For example:

```
option vendor-encapsulated-options
  2:4:AC:11:41:1:
  3:12:73:75:6e:64:68:63:70:2d:73:65:72:76:65:72:31:37:2d:31:
  4:12:2f:65:78:70:6f:72:74:2f:72:6f:6f:74:2f:69:38:36:70:63;
```

The second way of setting the value of this option is to have the DHCP server generate a vendor-specific option buffer. To do this, you must do four things: define an option space, define some options in that option space, provide values for them, and specify that that option space should be used to generate the **vendor-encapsulated-options** option.

To define a new option space in which vendor options can be stored, use the option space statement:

option space *name* **;**

The name can then be used in option definitions, as described earlier in this document. For example:

```
option space SUNW;
option SUNW.server-address code 2 = ip-address;
option SUNW.server-name code 3 = text;
option SUNW.root-path code 4 = text;
```

Once you have defined an option space and the format of some options, you can set up scopes that define values for those options, and you can say when to use them. For example, suppose you want to handle two different classes of clients. Using the option space definition shown in the previous example, you can send different option values to different clients based on the vendor-class-identifier option that the clients send, as follows:

```
class "vendor-classes" {
  match option vendor-class-identifier;
}

option SUNW.server-address 172.17.65.1;
option SUNW.server-name "sundhcp-server17-1";

subclass "vendor-classes" "SUNW.Ultra-5_10" {
```

```
  vendor-option-space SUNW;
  option SUNW.root-path "/export/root/sparc";
}

subclass "vendor-classes" "SUNW.i86pc" {
  vendor-option-space SUNW;
  option SUNW.root-path "/export/root/i86pc";
}
```

As you can see in the preceding example, regular scoping rules apply, so you can define values that are global in the global scope, and only define values that are specific to a particular class in the local scope. The **vendor-option-space** declaration tells the DHCP server to use options in the SUNW option space to construct the **vendor-encapsulated-options** option.

SEE ALSO

dhcpd.conf(5), dhcpd.leases(5), dhclient.conf(5), dhcp-eval(5), dhcpd(8), dhclient(8), RFC2132, RFC2131, draft-ietf-dhc-agent-options-??.txt.

AUTHOR

The Internet Systems Consortium DHCP Distribution was written by Ted Lemon under a contract with Vixie Labs. Funding for this project was provided through Internet Systems Consortium. Information about Internet Systems Consortium can be found at **http://www.isc.org.**

Ì¾¾Î

dhcp-options - Æ°Åª¥Û¥¹¥È¹½À®¥×¥í¥È¥³¥ë¤Î¥ª¥×¥·¥ç¥ó

²òÀâ

Æ°Åª¥Û¥¹¥È¹½À®¥×¥í¥È¥³¥ë (DHCP: Dynamic Host Configuration Protocol) ¤ò»ÈÍÑ¤¹¤ë¤³¤È¤Ë¤è¤ê¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ï DHCP ¥µ¡¼¥Ð¤«¤é¡¢¥Í¥Ã¥È¥ï¡¼¥¯ÀßÄê¤ä¥Í¥Ã¥È¥ï¡¼¥¯¾å¤ÇÍøÍÑ²ÄÇ½¤ÊÍÍ¡¹¤Ê¥µ¡¼¥Ó¥¹¤Ë¤Ä¤¤¤ÆµÒ½Ò¤·¤Æ¤¤¤ë **¥ª¥×¥·¥ç¥ó** ¤ò¼õ¤±¼è¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ **dhcpd(8)** ¤ä **dhclient(8)** ¤òÀßÄê¤¹¤ë¤È¤¤Ë¤Ï¡¢¤·¤Ð¤·¤Ð¥ª¥×¥·¥ç¥ó¤òÀë¸À¤¹¤ëÉ¬Í×¤¬¤¢¤ë¤Ç¤·¤ç¤¦¡£ ¤³¤³¤Ç¤Ï¡¢¥ª¥×¥·¥ç¥ó¤òÀë¸À¤¹¤ëÊ¸Ë¡¡¢¤½¤·¤ÆÀë¸À²ÄÇ½¤Ê¥ª¥×¥·¥ç¥ó¤Î̾Á°¤È½ñ¼°¤òÊ¸½ñ²½¤·¤Æ¤¤¤Þ¤¹¡£

¥ê¥Õ¥¡¥ì¥ó¥¹: ¥ª¥×¥·¥ç¥óÊ¸

DHCP *option* Ê¸¤Ï¡¢¾ï¤Ë¥¡¼¥ï¡¼¥É *option* ¤Ç³«»Ï¤·¡¢Ã±°ì¤Î¥ª¥×¥·¥ç¥ó̾¤¬Â³¤¡¢¥ª¥×¥·¥ç¥ó¥Ç¡¼¥¿¤¬Â³¤¤Þ¤¹¡£ ¥ª¥×¥·¥ç¥ó¤Î̾Á°¤È¥Ç¡¼¥¿¤Î½ñ¼°¤Ï¸å½Ò¤·¤Þ¤¹¡£ ¤¹¤Ù¤Æ¤Î DHCP ¥ª¥×¥·¥ç¥ó¤òÌÖÍåÅª¤Ë»ØÄê¤¹¤ëÉ¬Í×¤Ï¤Ê¤¯¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤ËÉ¬Í×¤Ê¥ª¥×¥·¥ç¥ó¤Î¤ß¤ò»ØÄê¤·¤Þ¤¹¡£

¥ª¥×¥·¥ç¥ó¥Ç¡¼¥¿¤Ë¤Ï¡¢¼¡¤Î¤è¤¦¤ËÍÍ¡¹¤Ê½ñ¼°¤¬¤¢¤ê¤Þ¤¹:

ip-address ¥Ç¡¼¥¿¥¿¥¤¥×¤Ï¡¢ÌÀ¼¨Åª¤Ê IP ¥¢¥É¥ì¥¹ (Îã¤¨¤Ð 239.254.197.10) ¤Þ¤¿¤Ï ¥É¥á¥¤¥ó̾ (Îã¤¨¤Ð haagen.isc.org) ¤Î¤É¤Á¤é¤Ç¤â»ØÄê²ÄÇ½¤Ç¤¹¡£ ¥É¥á¥¤¥ó̾¤Ç»ØÄê¤¹¤ë¾ì¹ç¡¢¤½¤Î¥É¥á¥¤¥ó̾¤ò²ò·è¤¹¤ë¤ÈÃ±°ì¤Î IP ¥¢¥É¥ì¥¹¤Ë¤Ê¤ë¤è¤¦¤Ë¤·¤Æ¤¯¤À¤µ¤¤¡£

int32 ¥Ç¡¼¥¿¥¿¥¤¥×¤ÏÉä¹æÉÕ¤ 32 ¥Ó¥Ã¥ÈÀ°¿ô¤ò»ØÄê¤·¤Þ¤¹¡£ **uint32** ¥Ç¡¼¥¿¥¿¥¤¥×¤ÏÉä¹æÌµ¤· 32 ¥Ó¥Ã¥ÈÀ°¿ô¤ò»ØÄê¤·¤Þ¤¹¡£ **int16** ¤ª¤è¤Ó **uint16** ¤Î¥Ç¡¼¥¿¥¿¥¤¥×¤Ï¡¢Éä¹æÉÕ¤¤ª¤è¤ÓÉä¹æÌµ¤·¤Î 16 ¥Ó¥Ã¥ÈÀ°¿ô¤ò»ØÄê¤·¤Þ¤¹¡£ **int8** ¤ª¤è¤Ó **uint8** ¤Î¥Ç¡¼¥¿¥¿¥¤¥×¤Ï¡¢Éä¹æÉÕ¤¤ª¤è¤ÓÉä¹æÌµ¤·¤Î 8 ¥Ó¥Ã¥ÈÀ°¿ô¤ò»ØÄê¤·¤Þ¤¹¡£ Éä¹æÌµ¤· 8 ¥Ó¥Ã¥ÈÀ°¿ô¤Ï¡¢¥ª¥¯¥Æ¥Ã¥È¤È¸Æ¤Ð¤ì¤ë¤³¤È¤â¤¢¤ê¤Þ¤¹¡£

text ¥Ç¡¼¥¿¥¿¥¤¥×¤Ï NVT ASCII Ê¸»úÎó¤ò»ØÄê¤·¤Þ¤¹¡£ Ê¸»úÎó¤Ï¥À¥Ö¥ë¥¯¥©¡¼¥È¤Ç³ç¤ëÉ¬Í×¤¬¤¢¤ê¤Þ¤¹¡£ Îã¤¨¤Ð root-path ¥ª¥×¥·¥ç¥ó¤ò»ØÄê¤¹¤ëÊ¸Ë¡¤Ï¡¢¼¡¤Î¤è¤¦¤Ë¤Ê¤ê¤Þ¤¹¡£

```
option root-path "10.0.1.4:/var/tmp/rootfs";
```

domain-name ¥Ç¡¼¥¿¥¿¥¤¥×¤Ï¥É¥á¥¤¥ó̾¤ò»ØÄê¤·¤Þ¤¹¡£ Ê¸»úÎó¤ò¥À¥Ö¥ë¥¯¥©¡¼¥È¤Ç³ç¤Ã¤Æ¤Ï¤¤¤±¤Þ¤»¤ó¡£ ¤³¤Î¥Ç¡¼¥¿¥¿¥¤¥×¤Ï¡¢Â¾¤Î´ûÂ¸¤Î DHCP ¥ª¥×¥·¥ç¥ó¤Ë¤Ï»È¤ï¤ì¤Þ¤»¤ó¡£ ¥É¥á¥¤¥ó̾¤Ï¡¢text ¥ª¥×¥·¥ç¥ó¤Ç¤¢¤ë¤«¤Î¤è¤¦¤ËÊÝ»ý¤µ¤ì¤Þ¤¹¡£

flag ¥Ç¡¼¥¿¥¿¥¤¥×¤Ï¥Ö¡¼¥ëÃÍ¤ò»ØÄê¤·¤Þ¤¹¡£ ¥Ö¡¼¥ëÃÍ¤Ï true ¤Þ¤¿¤Ï false ¤Î¤¤¤º¤ì¤«¤Ç¤¹ (¤â¤·¤¯¤Ï¡¢on ¤Þ¤¿¤Ï of **f** ¤ÎÊý¤¬Ê¬¤«¤ê¤ä¤¹¤±¤ì¤Ð¡¢¤³¤Á¤é¤Ç¤â¤«¤Þ¤¤¤Þ¤»¤ó)¡£

string ¥Ç¡¼¥¿¥¿¥¤¥×¤Ï¡¢¥À¥Ö¥ë¥¯¥©¡¼¥È¤Ç³ç¤é¤ì¤ë NVT ASCII Ê¸»úÎó¤«¡¢ ¥³¥í¥ó¶èÀÚ¤ê¤Î 16 ¿Ê¿ô¤Ç»ØÄê¤µ¤ì¤ë¥ª¥¯¥Æ¥Ã¥È¤ÎÏ¢Â³¤Î¤¤¤º¤ì¤«¤ò»ØÄê¤·¤Þ¤¹¡£ Îã¤¨¤Ð¼¡¤Î¤è¤¦¤Ë¤Ê¤ê¤Þ¤¹:

```
  option dhcp-client-identifier "CLIENT-FOO";
```

¤â¤·¤¯¤Ï

```
  option dhcp-client-identifier 43:4c:49:45:54:2d:46:4f:4f;
```

¼°¤òÍÑ¤¤¤¿¥ª¥×¥·¥ç¥óÃÍ¤ÎÀßÄê

¥¯¥é¥¤¥¢¥ó¥È¤¬Á÷½Ð¤¹¤ë¤¤¤¯¤Ä¤«¤Î¾ðÊó¤ò¡¢DHCP ¥ª¥×¥·¥ç¥ó¤Î¾ðÊó¤òÀßÄê¤¹¤ë¤Î¤Ë»È¤¨¤ë¤ÈÊØÍø¤Ê¾ì¹ç¤¬¤¢¤ê¤Þ¤¹¡£ ¤³¤ì¤ò¤¹¤ë¤Ë¤Ï¼°¤ÎÉ¾²Á¤¬ÍøÍÑ¤Ç¤¤Þ¤¹¡£ **dhcp-eval(5)** ¥Þ¥Ë¥å¥¢¥ë¥Ú¡¼¥¸¤Ë¼°¤Î½ñ¤Êý¤¬½Ò¤Ù¤é¤ì¤Æ¤¤¤Þ¤¹¡£ É¾²Á¤Î·ë²Ì¤ò¥ª¥×¥·¥ç¥ó¤ËÂåÆþ¤¹¤ë¤Ë¤Ï¡¢¥ª¥×¥·¥ç¥ó¤ò¼¡¤Î¤è¤¦¤ËÄêµÁ¤·¤Þ¤¹:

option *my-option* = *expression* **;**

Îã¤¨¤Ð¼¡¤Î¤è¤¦¤Ë¤·¤Þ¤¹:

```
  option hostname = binary-to-ascii (16, 8, "-",
                                    substring (hardware, 1, 6));
```

É¸½à DHCP ¥ª¥×¥·¥ç¥ó

¼¡¤Ë¼¨¤¹ÍÍ¡¹Ê¥ª¥×¥·¥ç¥ó¤Ë´Ø¤¹¤ëµ½Ò¤Ï¡¢ DHCP ¥ª¥×¥·¥ç¥ó¤Ë´Ø¤¹¤ëºÇ¿·¤Î IETF ¥É¥é¥Õ¥ÈÊ¸½ñ¤«¤é¤Î¤â¤Î¤Ç¤¹¡£
Ì¾Á°¤¬·ÇºÜ¤µ¤ì¤Æ¤¤¤Ê¤ª¥×¥·¥ç¥ó¤Ï¡¢¤Þ¤À¼ÂÁõ¤µ¤ì¤Æ¤¤¤Ê¤¤¤«¤â¤·¤ì¤Þ¤»¤ó¤¬¡¢
ÀßÄê¥Õ¥¡¤¥ë¤ËÄêµÁ¤¹¤ë¤³¤È¤Ç¡¢¤½¤Î¤è¡¦¤Ê¥ª¥×¥·¥ç¥ó¤ò»È¤¨¤ë¤«¤â¤·¤ì¤Þ¤»¤ó¡£
¾Ü¤·¤¯¤Ï¡¢¤³¤ÎÀè¤Î¡Ö¿·µ¬¥ª¥×¥·¥ç¥ó¤ÎêµÁ¡×¤«¤éÂ³¤¯µ½Ò¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

¤³¤³¤Ëµ½Ò¤µ¤ì¤Æ¤¤¤ë¥ª¥×¥·¥ç¥ó¤Î¡¦Á¤Î¤¤¯¤Ä¤«¤Ï¡¢DHCP ¥µ¡¼¥Ð¤â¤·¤¯¤Ï
¥¯¥é¥¤¥¢¥ó¥È¤Ë¤è¤Ã¤Æ¼«Æ°Å³¤ËÀ¸À®¤µ¤ì¤ë¤â¤Î¤Ç¡¢¥æ¡¼¥¶¤Ë¤ÏÀßÄê¤Ç¤¤Þ¤»¤ó¡£
¤½¤Î¤è¡¦¤Ê¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤Ï¡¢¼õ¿®Â¦¤Î DHCP ¥×¥í¥È¥³¥ë¥¨¡¼¥¸¥§¥ó¥È
(¥µ¡¼¥Ð¤â¤·¤¯¤Ï¥¯¥é¥¤¥¢¥ó¥È) ¤ÎÀßÄê¥Õ¥¡¤¥ë¤Æ¤Î¡¢¤Îã¤¨¤Ð¾ò·ï¼°¤Ê¤É¤Ç »È¤ï¤ì¤Þ¤¹¡£
¤·¤«¤·¤³¤Î¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤Ï¡¢Á÷¿®Â¦¤Î¥¨¡¼¥¸¥§¥ó¥È¤ÎÀßÄê¥Õ¥¡¤¥ë¤Æ¤Ç¤Ï
»È¤ï¤ì¤ë¤³¤È¤Ï¤¢¤ê¤Þ¤»¤ó¡£
¤È¤¤¤¦¤Î¤â¡¢¤½¤ÎÃÍ¤Ï¡¢ÀßÄê¥Õ¥¡¤¥ë¤¬¤½èÍý¤µ¤ì¤¿¸å¤Ë·èÄê¤µ¤ì¤ë¤«¤é¤Ç¤¹¡£ °Ê¹ß¤Î¡¢½Ò¤Ë¤ª¤¤¤Æ¡¢¤½¤Î¤è¡¦¤Ê¥ª¥×¥·¥ç¥ó¤Ë¤Ï ¡Ö¥æ¡¼¥¶¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤Þ¤»¤ó¡×¤È¡µ¤ì¤Þ¤¹¡£

É¸½à¥ª¥×¥·¥ç¥ó¤ò¼¨¤·¤Þ¤¹:

option all-subnets-local *flag*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬ÀÜÂ³¤µ¤ì¤Æ¤¤¤ë IP ¥Í¥Ã¥È¥ï¡¼¥¯¤ÎÁ´¥µ¥Ö¥Í¥Ã¥È¤¬ »ÈÍÑ¤¹¤ë MTU ¤¬¡¢¯¥é¥¤¥¢¥ó¥È¤¬Ä¾ÀÜÀÜÂ³¤µ¤ì¤Æ¤¤¤ë¥µ¥Ö¥Í¥Ã¥È¤Î MTU ¤È Æ±¤¸¤Ç¤¢¤È¡¢¯¥é¥¤¥¢¥ó¥È¤¬²¾Äê¤·¤Æ¤è¤¤¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ true ¤Ï¡¢Á´¥µ¥Ö¥Í¥Ã¥È¤ÏÆ±¤ì¤Î MTU ¤Ç¤¢¤ë³È¤ò°ÕÌ£¤·¤Þ¤¹¡£ ÃÍ false ¤Ï¡¢Ä¾ÀÜÀÜÂ³¤µ¤ì¤Æ¤¤¤ë¥Í¥Ã¥È¥ï¡¼¥¯¤Î¥µ¥Ö¥Í¥Ã¥È¤Ë¤Ï¡¢¤è¤ê¾®¤µ¤Ê MTU ¤ò »ý¤Ä¤â¤Î¤¬¤¢¤ë¤È¡¢¯¥é¥¤¥¢¥ó¥È¤¬²¾Äê¤¹¤Ù¤Ç¤¢¤ë³È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option arp-cache-timeout *uint32*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢ARP ¥¥ã¥Ã¥·¥å¥¨¥ó¥È¥ê¤Î¥¿¥¤¥à¥¢¥¦¥È¤òÉÃ¿ô¤Ç»ØÄê¤·¤Þ¤¹¡£

option bootfile-name *text*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢µ¯Æ°¶Õ¥¡¤¥ë¤ò»ØÄê¤¹¤ë¤¿¤á¤Ë»ÈÍÑ¤·¤Þ¤¹¡£
¥¯¥é¥¤¥¢¥ó¥È¤Ë¤è¤Ã¤Æ¥µ¥Ý¡¼¥È¤µ¤ì¤Æ¤¤¤ë¾ì¹ç¡¢ ¤³¤ì¤Ï **filename** Àë¸À¤È Æ±¤¸¤ú²Ì¤ò»ý¤Á¤Þ¤¹¡£ BOOTP ¥¯¥é¥¤¥¢¥ó¥È¤Ç¡¢¤³¤Î¥ª¥×¥·¥ç¥ó¤ò¥µ¥Ý¡¼¥È¤·¤Æ¤¤¤ë¤â¤Î¤¯¤Ê¤¤¤Ç¤·¤ç¤¦¡£ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ë¤è¤Ã¤Æ¤Ï¥µ¥Ý¡¼¥È¤¤¤ë¤â¤Î¤¬¤¢¤ê¡¢ ¼ÂºÝÉé¬¿Ü¤È¤·¤Æ¤¤¤ë¤â¤Î¤¬¤¢¤ê¤Þ¤¹¡£

option boot-size *uint16*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥ÈÍÑ¤Î¥Ç¥Õ¥©¥ë¥È¤Î¥Ö¡¼¥È¥¤á¡¼¥¸¤ÎÄ¹¤µ¤ò¡¢ 512 ¥ª¯¥Æ¥Ã¥È¥Ö¥í¥Ã¥¯¿ô¤Ç»ØÄê¤·¤Þ¤¹¡£

option broadcast-address *ip-address*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Î¥µ¥Ö¥Í¥Ã¥È¤Ç»ÈÍÑ¤µ¤ì¤Æ¤¤¤ë ¥Ö¥í¡¼¥É¥-¥ã¥¹¥È¥¢¥É¥ì¥¹¤ò»ØÄê¤·¤Þ¤¹¡£ ÀµÅö¤Ê¥Ö¥í¡¼¥É¥¥ã¥¹¥È¥¢¥É¥ì¥¹¤ÎÃÍ¤Ï¡¢STD 3 (RFC1122) ¤Î 3.2.1.3 Àá¤Ë µ¬Äê¤µ¤ì¤Æ¤¤¤Þ¤¹¡£

option cookie-servers *ip-address* [**,** *ip-address...*]**;**

¥¯¥Ã¥¡¼¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê RFC 865 ¥¯¥Ã¥-¡¼¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option default-ip-ttl *uint8;*

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥Ç¡¼¥¿¥¿¥°¥é¥à¤òÁ÷½Ð¤¹¤ë¤È¤¤Ë»ÈÍÑ¤¹¤Ù¤¡¢ ¥Ç¥Õ¥©¥ë¥È¤ÎÀ¸Â¸»þ´Ö (TTL) ¤ò»ØÄê¤·¤Þ¤¹¡£

option default-tcp-ttl *uint8*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬ TCP ¥»¥°¥á¥ó¥È¤òÁ÷½Ð¤¹¤ë¤È¤¤Ë»ÈÍÑ¤¹¤Ù¤¡¢ ¥Ç¥Õ¥©¥ë¥È¤Î TTL ¤ò»ØÄê¤·¤Þ¤¹¡£ ºÇ¾®ÃÍ¤Ï 1 ¤Ç¤¹¡£

option dhcp-client-identifier *string*;

ËÜ¥ª¥×¥·¥ç¥ó¤ò»È¤Ã¤Æ¡¢¥Û¥¹¥ÈÀë¸ÀÃæ¤Ç DHCP ¥¯¥é¥¤¥¢¥ó¥È¼±ÊÌ»Ò¤ò »ØÄê¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹£ ¤³¤Î¥¯¥é¥¤¥¢¥ó¥È¼±ÊÌ»Ò¤Ç¾Ê¹ç¤ò¹Ô¤¦¤³¤È¤Ç¡¢ dhcpd ¤Ï¤½Î¥Û¥¹¥È¤Î¥ì¾¡¼¥É¤òÈ¯«¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹£

DHCP ¥¯¥é¥¤¥¢¥ó¥È¤ÎÃæ¤Ë¤Ï¡¢ASCII ¥Æ¥¥¹¥È¤Ë¤è¤Ã¤Æ¥¯¥é¥¤¥¢¥ó¥È¼±ÊÌ»Ò¤¬ ÀßÄê¤µ¤ì¤¿¾ì¹ç¡¢¤½¤Î ASCII ¥Æ¥¥¹¥È¤ÎÀèÆ¬¤Ë 0 ¤ò¤Ä¤±¤ë¤â¤Î¤¬¤¢¤ë¤³¤È¤Ë Ãí°Õ¤·¤Æ¤¯¤À¤µ¤¤¡£ ¤½¤Î¾ì¹ç¡¢

option dhcp-client-identifier "foo";

¤Ç¤Ï¤Ê¤¯¡¢°Ê²¼¤Î¤è¤¦¤Ëµ½Ò¤¹¤ëÉ¬Í×¤¬¤¢¤ë¤Ç¤·¤ç¤¦¡£

option dhcp-client-identifier "\0foo";

option dhcp-lease-time *uint32***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥ÈÍ×µá (DHCPDISCOVER ¤Þ¤¿¤Ï DHCPREQUEST) ¤ÎÃæ¤Ç¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬ IP ¥¢¥É¥ì¥¹Î¥ê¡¼¥¹»þ´Ö¤òÍ×µá¤¹¤ë¤¿¤á¤Ë»ÈÍÑ¤µ¤ì¤Þ¤¹£ ¤Þ¤¿¥µ¡¼¥Ð±þÅú (DHCPOFFER) ¤ÎÃæ¤Ç¡¢DHCP ¥µ¡¼¥Ð¤¬Äó¼¨¤·¤¿¤¤ê¡¼¥¹»þ´Ö¤ò »ØÄê¤¹¤ë¤Î¤Ë¤â¡¢¤³¤Î¥ª¥×¥·¥ç¥ó¤Ï»È¤ï¤ì¤Þ¤¹£

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥µ¡¼¥Ð¤Ç¤Ï¥æ¡¼¥¶¤¬Ä¾ÀÜÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤Þ¤»¤ó¡£ **dhcpd.conf(5)** ¤Î *max-lease-time* ¤È default-lease-time ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤ò »²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

option dhcp-max-message-size *uint16***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤«¤éÁ÷½Ð¤µ¤ì¤¿¾ì¹ç¡¢¥µ¡¼¥Ð¤¬¯¥é¥¤¥¢¥ó¥È¤Ë Á÷½Ð¤¹¤ë¤¹Ù¤Æ¤Î±þÅú¤ÎºÇÂç¥µ¥¤¥ºò»ØÄê¤·¤Þ¤¹£ ¥µ¡¼¥Ð¤ÇÀßÄê¤µ¤ì¤¿¾ì¹ç¡¢¯¥é¥¤¥¢¥ó¥È¤¬ dhcp-max-message-size ¥ª¥×¥·¥ç¥ó¤ò Á÷¿®¤·¤Æ¤³¤Ê¤«¤Ã¤¿°Ý¤Ë¡¢¤³¤Î¥µ¡¼¥Ð¤ÇÀßÄê¤µ¤ì¤¿¤ÃÍ¤¬»ÈÍÑ¤µ¤ì¤Þ¤¹£ ¤³¤ì¤Ï¡¢BOOTP ±þÅú¤Ç¤â DHCP ±þÅú¤ÈÆ±ÍÍ¤ËÆ°ºî¤·¤Þ¤¹£

option dhcp-message *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¾ã³²¤¬µ¯¤¤¿»þ¤Ë¡¢DHCP ¥µ¡¼¥Ð¤¬ DHCPNAK ¥á¥Ã¥»¡¼¥¸Ãæ¤Ç DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ø¥¨¥é¡¼¥á¥Ã¥»¡¼¥¸¤òÄó¶¡¤¹¤ë¤Ë»ÈÍÑ¤·¤Þ¤¹£ ¤Þ¤¿¥¯¥é¥¤¥¢¥ó¥È¤¬¡¢ÄóµÜ¤µ¤ì¤¿¥Ñ¥é¥á¡¼¥¿¤òµñÈÝ¤·¤¿¾ÍýÍ³¤ò¼¨¤¹¤¿¤á¤Ë¡¢ DHCPDECLINE ¥á¥Ã¥»¡¼¥¸Ãæ¤ÇËÜ¥ª¥×¥·¥ç¥ó¤ò»È¤¦¤³¤È¤â¤ê¤Þ¤¹£

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤Þ¤»¤ó¡£

option dhcp-message-type *uint8***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¥µ¡¼¥ÐÎ¾¼Ô¤«¤éÁ÷½Ð¤µ¤ì¡¢ DHCP ¥Ñ¥±¥Ã¥È¤´Þ¤ó¤Ç¤¤¤ë DHCP ¥á¥Ã¥»¡¼¥¸¸¡¤Î¥¿¡¼¥×¥×¤¤»ØÄê¤·¤Þ¤¹£ ËÜ¥ª¥×¥·¥ç¥ó¤¬¼è¤êÀÅ¤ëÃÍ¤Ï¡¢°Ê²¼¤Î¤è¤ª¤ê¤Ç¤¹ (RFC2132 ¤è¤ê¤½¤Î¤Þ¤ÞÈ´¿è)¡£

1 DHCPDISCOVER
2 DHCPOFFER
3 DHCPREQUEST
4 DHCPDECLINE
5 DHCPACK
6 DHCPNAK
7 DHCPRELEASE
8 DHCPINFORM

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤Þ¤»¤ó¡£

option dhcp-option-overload *uint8***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢DHCP ’sname’ ¤â¤·¤¯¯¤Ï ’file’ ¥Õ¥£¡¼¥ë¥É¤¬¡¢ DHCP ¥ª¥×¥·¥ç¥ó¤ÒÝ»ý¤¤¤ë¤¿¤á¤ËµÍ¤á¹Þ¤ß²á¤®¤Ë¤Ê¤Ã¤Æ¤¤¤ë¤³¤È¤ò ¼¨¤¹¤Î¤Ë»È¤¤¤¿¤Þ¤Þ¤¹£ DHCP ¥µ¡¼¥Ð¤¬¡¢ÊÖµÑ¤µ¤ì¤¿¥Ñ¥é¥á¡¼¥¿¤¬¡¢ª¥×¥·¥ç¥ó¤ËÄÌ³ï³ä¤êÅö¤Æ¤é¤ì¤¿

¶õ´Ö¤òÄ¶²á¤·¤¿¾ì¹ç¡¢ËÜ¥ª¥×¥·¥ç¥ó¤òÁÞÆþ¤·¤Þ¤¹¡£

ËÜ¥ª¥×¥·¥ç¥ó¤¬Â¸ºß¤·¤¿¾ì¹ç¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢É¸½à¤Î¥ª¥×¥·¥ç¥ó¥Õ¥£¡¼¥ë¥É¤Î ²ò¼á¤¬½ªÎ»¤·¤¿¸å¡¢»ØÄê¤µ¤ì¤¿¥Õ¥£¡¼¥ë¥É¤Î²ò¼á¤ò¹Ô¤¤¤Þ¤¹¡£

ËÜ¥ª¥×¥·¥ç¥ó¤ÎÀµÅö¤ÊÃÍ¤Ï¡¢°Ê²¼¤ÎÄÌ¤ê¤Ç¤¹:

1 ’file’ ¥Õ¥£¡¼¥ë¥É¤¬¡¢¥ª¥×¥·¥ç¥óÊÝ»ý¤Ë»ÈÍÑ¤µ¤ì¤Æ¤Þ¤¹
2 ’sname’ ¥Õ¥£¡¼¥ë¥É¤¬¡¢¥ª¥×¥·¥ç¥óÊÝ»ý¤Ë»ÈÍÑ¤µ¤ì¤Æ¤Þ¤¹
3 Î¾Êý¤Î¥Õ¥£¡¼¥ë¥É¤¬¡¢¥ª¥×¥·¥ç¥óÊÝ»ý¤Ë»ÈÍÑ¤µ¤ì¤Æ¤Þ¤¹

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤¤Þ¤»¤ó¡£

option dhcp-parameter-request-list *uint16*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤«¤éÁ÷½Ð¤µ¤ì¤¿¾ì¹ç¡¢ ¥µ¡¼¥Ð¤ËÊÖÅú¤ò´õË¾¤¹¤ë¥ª¥×¥·¥ç¥ó¤ò¥¯¥é¥¤¥¢¥ó¥È¤¬»ØÄê¤·¤Þ¤¹¡£ ÄÌ¾ï ISC DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ç¤Ï¡¢*request* Ê¸¤òÍÑ¤¤¤ÆÆ¹Ô¤ï¤ì¤Þ¤¹¡£ ËÜ¥ª¥×¥·¥ç¥ó¤¬¥¯¥é¥¤¥¢¥ó¥È¤«¤é»ØÄê¤µ¤ì¤Ê¤«¤Ã¤¿¾ì¹ç¡¢ÄÌ¾ï DHCP ¥µ¡¼¥Ð¤Ï¡¢ ¥¹¥³¡¼¥×Æâ¤ÇÍ¸ú¤«¤Ä±þÅú¤Ë¼ý¤Þ¤ë¤¹¤Ù¤Æ¤Î¥ª¥×¥·¥ç¥ó¤òÊÖ¤·¤Þ¤¹¡£ ËÜ¥ª¥×¥·¥ç¥ó¤¬¥µ¡¼¥Ð¾å¤Ç»ØÄê¤µ¤ì¤¿¾ì¹ç¡¢¥µ¡¼¥Ð¤Ï¤½¤Î»ØÄê¤µ¤ì¤¿¥ª¥×¥·¥ç¥ó¤ò ÊÖ¤·¤Þ¤¹¡£ ¤³¤ì¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬Í×µá¤·¤Ê¤«¤Ã¤¿¥ª¥×¥·¥ç¥ó¤ò¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ë ¶¯À©¤¹¤ë¤Î¤Ë»ÈÍÑ¤µ¤ì¤Þ¤¹¡£ ¤Þ¤¿¡¢ÄÌ¾ï¥µ¡¼¥Ð¤¬ÊÖ¤¹¥ª¥×¥·¥ç¥ó¤Î¥»¥Ã¥È¤ò¤µ¤é¤ËÀ©¸Â¤¹¤ëÉ¬Í×¤Î¤¢¤ë ¥¯¥é¥¤¥¢¥ó¥È¤ËÂÐ¤·¤Æ¡¢DHCP ¥µ¡¼¥Ð¤Î±þÅú¤òÄ´À°¤¹¤ë¤Î¤Ë¤â»ÈÍÑ¤µ¤ì¤Þ¤¹¡£

option dhcp-rebinding-time *uint32*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬¡¢¥¢¥É¥ì¥¹¤ò¼èÆÀ¤·¤Æ¤«¤é REBINDING ¾õÂÖ¤Ë °Ü¹Ô¤¹¤ë¤Þ¤Ç¤Î»þ´Ö¤òÉÃ¿ô¤Ç»ØÄê¤·¤Þ¤¹¡£

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤¤Þ¤»¤ó¡£

option dhcp-renewal-time *uint32*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬¡¢¥¢¥É¥ì¥¹¤ò¼èÆÀ¤·¤Æ¤«¤é RENEWING ¾õÂÖ¤Ë °Ü¹Ô¤¹¤ë¤Þ¤Ç¤Î»þ´Ö¤òÉÃ¿ô¤Ç»ØÄê¤·¤Þ¤¹¡£

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤¤Þ¤»¤ó¡£

option dhcp-requested-address *ip-address*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬¡¢DHCPDISCOVER Æâ¤ÇÆÃÄê¤Î IP ¥¢¥É¥ì¥¹¤¬ ³ä¤êÅö¤Æ¤é¤ì¤ë¤³¤È¤òÍ×µá¤¹¤ë¤Î¤Ë»ÈÍÑ¤µ¤ì¤Þ¤¹¡£

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤¤Þ¤»¤ó¡£

option dhcp-server-identifier *ip-address*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢DHCPOFFER ¤È DHCPREQUEST ¥á¥Ã¥»¡¼¥¸Ãæ¤Ç»ÈÍÑ¤µ¤ì¡¢ ¤Þ¤¿ DHCPACK ¤È DHCPNAK ¥á¥Ã¥»¡¼¥¸Ãæ¤Ë¤â´Þ¤Þ¤ì¤ë¤³¤È¤¬¤¢¤ê¤Þ¤¹¡£ DHCP ¥µ¡¼¥Ð¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ (ÌõÃí: Ê£¿ô¥µ¡¼¥Ð¤«¤é¤Î) ¥ê¡¼¥¹¤ÎÄó¼¨¤ò ¶èÊÌ¤Ç¤¤ë¤è¤¦¡¢DHCPOFFER ¤ËËÜ¥ª¥×¥·¥ç¥ó¤ò´Þ¤á¤Þ¤¹¡£ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢DHCP ¥µ¡¼¥Ð¤Ø¥æ¥Ë¥¥ã¥¹¥È¤¹¤ë¤¹¤Ù¤Æ¤Î DHCP ¥á¥Ã¥»¡¼¥¸¤Î °¸Àè¥¢¥É¥ì¥¹¤È¤·¤Æ ’server identifier’ ¥Õ¥£¡¼¥ë¥É¤ÎÆâÍÆ¤ò»ÈÍÑ¤·¤Þ¤¹¡£ ¤Þ¤¿ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢DHCPREQUEST ¥á¥Ã¥»¡¼¥¸Ãæ¤ËËÜ¥ª¥×¥·¥ç¥ó¤ò´Þ¤á¡¢ Ê£¿ô¤Î¥ê¡¼¥¹¤ÎÄó¼¨¤Î¤É¤ì¤ò¼õ¤±Æþ¤ì¤¿¤«¤ò¼¨¤·¤Þ¤¹¡£

ËÜ¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤Ï¡¢¥µ¡¼¥Ð¤Î IP ¥¢¥É¥ì¥¹¤Ç¤¹¡£

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤¬Ä¾ÀÜÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤¤Þ¤»¤ó¡£ *dhcpd.conf(5)* ¤Î *server-identifier* ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

option domain-name *text*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥É¥á¥¤¥ó¥Í¡¼¥à¥·¥¹¥Æ¥à¤ò»ÈÍÑ¤·¤Æ¥Û¥¹¥È¾¤ò²ò·è¤¹¤ë¤È¤¤Ë ¥¯¥é¥¤¥¢¥ó¥È¤¬»ÈÍÑ¤¹¤Ù¤¥É¥á¥¤¥óÌ¾¤ò»ØÄê¤·¤Þ¤¹¡£

option domain-name-servers *ip-address* [**,** *ip-address...*]**;**

domain-name-servers ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê ¥É¥á¥¤¥ó¥Í¡¼¥à¥·¥¹¥Æ¥à (STD 13, RFC 1035) ¤Î¥Í¡¼¥à¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option extensions-path *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢ÄÉ²Ã¥ª¥×¥·¥ç¥ó¤ò´Þ¤à¥Õ¥¡¥¤¥ë¤Î¥Õ¥¡¥¤¥ëÌ¾¤ò»ØÄê¤·¤Þ¤¹¡£ ¤³¤ÎÄÉ²Ã¥ª¥×¥·¥ç¥ó¤Ï¡¢RFC2132 ¤Çµ¬Äê¤µ¤ì¤Æ¤¤¤ë DHCP ¥ª¥×¥·¥ç¥ó¤Î½ñ¼°¤Ë±è¤Ã¤Æ²ò¼á¤µ¤ì¤Þ¤¹¡£

option finger-server *ip-address* [**,** *ip-address...*]**;**

Finger ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê Finger ¤Î¥ê¥¹¥È¤ò »ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option font-servers *ip-address* [**,** *ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê X Window System ¥Õ¥©¥ó¥È¥µ¡¼¥Ð¤ò »ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option host-name *string***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤ÎÌ¾Á°¤ò»ØÄê¤·¤Þ¤¹¡£ ¤³¤ÎÌ¾Á°¤Ï¡¢¥í¡¼¥«¥ë¥É¥á¥¤¥óÌ¾¤Ë½¾¾þ¤µ¤ì¤Æ¤¤Æ¤â¡¢¤¤Ê¤¯¤â¤Þ¤¤¤»¤ó (¥É¥á¥¤¥óÌ¾¤ò»ØÄê¤¹¤ë¤Ï¡¢domain-name ¥ª¥×¥·¥ç¥ó¤Î»ÈÍÑ¤ò¤ª´«¤á¤·¤Þ¤¹)¡£ Ê¸»ú½¸¹ç¤ÎÀ©Ìó¤Ë¤Ä¤¤Æ¤Ï RFC 1035 ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£ ¥¯¥é¥¤¥¢¥ó¥È¥Þ¥·¥ó¤Î¥Û¥¹¥ÈÌ¾¤¬ÀßÄê¤µ¤ì¤Æ¤¤¤Ê¤¤¾ì¹ç (¤¹¤Ê¤ïÁ **rc.conf(5)** ¤Ç¶õÊ¸»úÎó¤ËÀßÄê¤µ¤ì¤Æ¤¤¤ë¾ì¹ç) ¤Î¤ß¡¢ **dhclient-script(8)** ¤¬ËÜ¥ª¥×¥·¥ç¥ó¤òÂº½Å¤·¤Þ¤¹¡£

option ieee802-3-encapsulation *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¤¥ó¥¿¾¥§¡¼¥¹¤¬¥¤¡¼¥µ¥Í¥Ã¥È¤Ç¤¢¤ë¾ì¹ç¤Ë¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬¥¤¡¼¥µ¥Í¥Ã¥È¥Ð¡¼¸¸¥ç¥ó 2 (RFC 894) ¤È IEEE 802.3 (RFC 1042) ¤Î¤É¤Á¤é¤Î¥«¥×¥»¥ë²½¤ò»ÈÍÑ¤¹¤Ù¤¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ false ¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ RFC 894 ¤Î¥«¥×¥»¥ë²½¤ò»ÈÍÑ¤¹¤Ù¤¤Ç¤¢¤ë¤³¤È¤ò °ÕÌ£¤·¤Þ¤¹¡£ ÃÍ true ¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ RFC 1042 ¤Î¥«¥×¥»¥ë²½¤ò»ÈÍÑ¤¹¤Ù¤¤Ç¤¢¤ë¤³¤È¤ò °ÕÌ£¤·¤Þ¤¹¡£

option ien116-name-servers *ip-address* [**,** *ip-address...*];

ien116-name-servers ¥ª¥×¥·¥ç¥ó¤Ï¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê IEN 116 ¥Í¡¼¥à¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option impress-servers *ip-address* [**,** *ip-address...*]**;**

impress-server ¥ª¥×¥·¥ç¥ó¤Ï¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê Imagen Impress ¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option interface-mtu *uint16***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¤³¤Î¥ó¥¿¾¥§¡¼¥¹¤¬¥¤¡¼¥µ¥Í¥Ã¥È¤Ç¤¢¤ë¾ì¹ç¤Ë¡¢ MTU ¤ò»ØÄê¤·¤Þ¤¹¡£ MTU ¤ËÂÐ¤¹¤ëºÇ¾®¤ÎÀµÅöÃÍ¤Ï 68 ¤Ç¤¹¡£

option ip-forwarding *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬¥¡¢¥Ñ¥±¥Ã¥È¤òÅ¾Á÷¤¹¤ë¤è¦¤Ë ¼«Ê¬¤Î IP ÁØ¤òÀßÄê¤¹¤Ù¤¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ false ¤Ï IP Å¾Á÷¤òÌµ¸ú¤Ë¤¹¤ë¤³¤È¤ò°ÕÌ£¤·¡¢ ÃÍ true ¤Ï IP Å¾Á÷¤òÍ-¸ú¤Ë¤¹¤ë¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option irc-server *ip-address* [**,** *ip-address...*]**;**

IRC ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê IRC ¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option log-servers *ip-address* [**,** *ip-address...*]**;**

log-server ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê MITLCS UDP ¥í¥°¥µ¡¼¥Ð¤Î ¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option lpr-servers *ip-address* [**,** *ip-address...*]**;**

LPR ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê RFC 1179 ¥é¥¤¥ó¥×¥ê¥ó¥¿¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£
¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option mask-supplier *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢ICMP ¤ò»ÈÍÑ¤·¤¿¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¥¯Í×µá¤ËÂÐ¤·¤Æ¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬±þÅú¤¹¤Ù¤¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ false ¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬±þÅú¤¹¤Ù¤-¤Ç¤Ê¤¤¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£ ÃÍ true ¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬±þÅú¤¹¤Ù¤¤Ç¤¢¤ë¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option max-dgram-reassembly *uint16***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ºÆÁÈ¤ßÎ©¤Æ¤Î½àÈ÷¤ò¤¹¤Ù¤
ºÇÂç¥Ç¡¼¥¿¥é¥à¥µ¥¤¥º¤ò»ØÄê¤·¤Þ¤¹¡£ ºÇ¾®¤ÎÀµÅöÃÍ¤Ï 576 ¤Ç¤¹¡£

option merit-dump *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬¥¯¥é¥Ã¥·¥å¤¹¤ë¤È¤¤Î
¥¯¥é¥¤¥¢¥ó¥È¤Î¥³¥¢¥¤á¡¼¸¤¬¥À¥ó¥×¤µ¤ì¤ë¥Õ¡¼¥ë¤Î¥Ñ¥¹Ì¾¤ò»ØÄê¤·¤Þ¤¹¡£
¥Ñ¥¹Î½ñ¼°¤Ï¡¢NVT ASCII Ê¸»ú½¸¹ç¤ÎÊ¸»ú¤«¤é¤Ê¤ëÊ¸»úÍó¤Ç¤¹¡£

option mobile-ip-home-agent *ip-address* [**,** *ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê¥â¥Ð¥¤¥ë IP ¥Û¡¼¥à¥¨¡¼¥¸¥§¥ó¥È¤Î IP ¥¢¥É¥ì¥¹Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£
¥¨¡¼¥¸¥§¥ó¥È¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£ ¤¿¤À¤·¡¢ÄÌ¾ï¥¨¡¼¥¸¥§¥ó¥È¤Ï 1 ¤Ä¤Ç¤·¤ç¤¦¡£

option nds-context *string***;**

nds-context ¥ª¥×¥·¥ç¥ó¤Ï¡¢NDS ¥¯¥é¥¤¥¢¥ó¥È¤Î¤¿¤á¤ÎºÇ½é¤Î NetWare ¥Ç¥£¥ì¥¯¥È¥ê¥µ¡¼¥Ó¥¹¤ÎÌ¾Á°¤ò»ØÄê¤·¤Þ¤¹¡£

option nds-servers *ip-address* [**,** *ip-address...*]**;**

nds-servers ¥ª¥×¥·¥ç¥ó¤Ï¡¢NDS ¥µ¡¼¥Ð¤Î IP ¥¢¥É¥ì¥¹Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£

option nds-tree-name *string***;**

nds-tree-name ¥ª¥×¥·¥ç¥ó¤Ï¡¢NDS ¥¯¥é¥¤¥¢¥ó¥È¤¬»ÈÍÑ¤¹¤Ù¤ NDS ¥Ä¥ê¡¼¤Î Ì¾Á°¤ò»ØÄê¤·¤Þ¤¹¡£

option netbios-dd-server *ip-address* [**,** *ip-address...*]**;**

NetBIOS ¥Ç¡¼¥¿¥é¥ÇÛÉÛ¥µ¡¼¥Ð (NBDD) ¥ª¥×¥·¥ç¥ó¤Ï¡¢ RFC 1001/1002 ¤Î NBDD ¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë»ØÄê¤·¤Þ¤¹¡£

option netbios-name-servers *ip-address* [**,** *ip-address...*]**;**

NetBIOS ¥Í¡¼¥à¥µ¡¼¥Ð (NBNS) ¥ª¥×¥·¥ç¥ó¤Ï¡¢ RFC 1001/1002 ¤Î NBNS ¥Í¡¼¥à¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë»ØÄê¤·¤Þ¤¹¡£ ¸½²ß¤Ç¤Ï¡¢NetBIOS ¥Í¡¼¥à¥µ¡¼¥Ó¥¹¤Ï WINS ¤È¸Æ¤Ð¤ì¤ë¤³¤È¤ÎÊý¤¬Â¿¤¤¤Ç¤¹¡£ netbios-name-servers ¥ª¥×¥·¥ç¥ó¤ò»ÈÍÑ¤·¤Æ¡¢WINS ¥µ¡¼¥Ð¤ò»ØÄê²ÄÇ½¤Ç¤¹¡£

option netbios-node-type *uint8***;**

NetBIOS ¥Î¡¼¥É¥¿¥×¥ª¥×¥·¥ç¥ó¤Ï¡¢ ÀßÄê²ÄÇ½¤Ê NetBIOS over TCP/IP ¥¯¥é¥¤¥¢¥ó¥È¤ò¡¢ RFC 1001/1002 ¤Ëµ½Ò¤µ¤ì¤Æ¤¤¤ë¤è¤¦¤ËÀßÄê¤·¤Þ¤¹¡£ ÃÍ¤ÏÃ±°ì¤Î¥ª¥Æ¥Ã¥È¤È¤·¤Æ»ØÄê¤µ¤ì¡¢¥¯¥é¥¤¥¢¥È¥¿¥×¥×¤ò°ÕÌ£¤·¤Þ¤¹¡£

»ÈÍÑ²ÄÇ½¤Ê¥Î¡¼¥É¥¿¥×¥×¤Ï¼¡¤ÎÀÌ¤ê¤Ç¤¹:

1 B ¥Î¡¼¥É: ¥Ö¥í¡¼¥É¥¥ã¥¹¥È - WINS Ìµ¤·

2 P ¥Î¡¼¥É: ¥Ô¥¢ - WINS ¤Î¤ß

4 M ¥Î¡¼¥É: ¥ß¥Ã¥¯¥¹ - ¥Ö¥í¡¼¥É¥¥ã¥¹¥È¸å¤Ë WINS

8 H ¥Î¡¼¥É: ¥Ï¥¤¥Ö¥ê¥Ã¥É - WINS ¸å¤Ë¥Ö¥í¡¼¥É¥¥ã¥¹¥È

option netbios-scope *string***;**

NetBIOS ¥¹¥³¡¼¥×¥ª¥×·¥ç¥ó¤Ï¡¢RFC 1001/1002 ¤Ëµ¬Äê¤µ¤ì¤Æ¤¤¤ë¤è¤¦¤Ë¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤Î NetBIOS over TCP/IP ¥¹¥³¡¼¥×¥Ñ¥é¥á¡¼¥¿¤ò»ØÄê¤·¤Þ¤¹¡£ Ê¸»ú½¸¹ç¤ÎÀ©Ìó¤Ë¤Ä¤¤Æ¤Ï RFC1001, RFC1002, RFC1035 ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

option nis-domain *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤Î NIS (Sun Network Information Services) ¥É¥á¥¤¥ó¤ò»ØÄê¤·¤Þ¤¹¡£ ¥É¥á¥¤¥ó¤Î½ñ¼°¤Ï¡¢NVT ASCII Ê¸»ú½¸¹ç¤ÎÊ¸»ú¤«¤é¤Ê¤ëÊ¸»úÎó¤Ç¤¹¡£

option nis-servers *ip-address* [**,** *ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê NIS ¥µ¡¼¥Ð¤ò¼¨¤¹ IP ¥¢¥É¥ì¥¹¤Î ¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option nisplus-domain *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤Î NIS+ ¥É¥á¥¤¥ó¤ÎÌ¾Á°¤ò»ØÄê¤·¤Þ¤¹¡£ ¥É¥á¥¤¥ó¤Î½ñ¼°¤Ï¡¢NVT ASCII Ê¸»ú½¸¹ç¤ÎÊ¸»ú¤«¤é¤Ê¤ëÊ¸»úÎó¤Ç¤¹¡£

option nisplus-servers *ip-address* [**,** *ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê NIS+ ¥µ¡¼¥Ð¤ò¼¨¤¹ IP ¥¢¥É¥ì¥¹¤Î ¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option nntp-server *ip-address* [**,** *ip-address...*]**;**

NNTP ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê NNTP ¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option non-local-source-routing *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢È¤ó¥í¡¼¥«¥ë»ØÄê·ÐÏ© (non-local source route) ¤ò»ý¤Ä ¥Ç¡¼¥¿¥°¥é¥à¤òÅ¾Á÷¤¹¤ë¤è¤¦¤Ë¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬¼«Ê¬¤Î IP ÁØ¤òÀßÄê¤¹¤Ù¤¤«¤ò »ØÄê¤·¤Þ¤¹ (ËÜ¹àÌÜ¤Ë¤Ä¤¤Æ¤Ï [4] ¤Î 3.3.5 Àá¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤)¡£ ÃÍ false ¤Ï¤½¤Î¤è¤¦¤Ê¥Ç¡¼¥¿¥°¥é¥à¤ÎÅ¾Á÷¤òµö²Ä¤·¤Ê¤¤¤³¤È¤ò°ÕÌ£¤·¡¢ ÃÍ true ¤ÏÅ¾Á÷µö²Ä¤ò°ÕÌ£¤·¤Þ¤¹¡£

option ntp-servers *ip-address* [**,** *ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê NTP (RFC 1035) ¥µ¡¼¥Ð¤ò¼¨¤¹ IP ¥¢¥É¥ì¥¹¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option nwip-domain *string***;**

NetWare/IP ¥¯¥é¥¤¥¢¥ó¥È¤¬»ÈÍÑ¤¹¤Ù¤ NetWare/IP ¥É¥á¥¤¥ó¤ÎÌ¾Á°¤Ç¤¹¡£

option nwip-suboptions *string***;**

NetWare/IP ¥¯¥é¥¤¥¢¥ó¥ÈÍÑ¤Î¥µ¥Ö¥ª¥×¥·¥ç¥ó¤Î¥·¡¼¥±¥ó¥¹¤Ç¤¹¡£ ¾Ü¤·¤¯¤Ï RFC2242 ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£ ÄÌ¾ï¡¢ËÜ¥ª¥×¥·¥ç¥ó¤ÏÆÃÃÄê¤Î NetWare/IP ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤ò»ØÄê¤¹¤ë¤³¤È¤Ç ÀßÄê¤µ¤ì¤Þ¤¹¡£ ¤µ¤é¤Ê¤ë¾ðÊó¤Ï¡ÖNetWare/IP ¥µ¥Ö¥ª¥×¥·¥ç¥ó¡×¤Î¾Ï¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

option path-mtu-aging-timeout *uint32***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢RFC 1191 ¤ÇÄêµÁ¤µ¤ì¤ë¡¹½¸¤ÇÈ¯¸«¤µ¤ì¤¿¥Ñ¥¹ MTU ÃÍ¤Î ¥¨¡¼¥¸¥§¥ó¥È»ÈÍÑ¤¹¤ë¥¿¥¤¥à¥¢¥¦¥È (ÉÃÃ±°Ì) ¤ò»ØÄê¤·¤Þ¤¹¡£

option path-mtu-plateau-table *uint16* [**,** *uint16...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢RFC 1191 ¤ÇÄêµÁ¤µ¤ì¤ë¥Ñ¥¹ MTU Ãµº÷ (Path MTU Discovery)

¼Â»Ü»þ¤Ë»ÈÍÑ¤µ¤ì¤ë MTU ¤Î¥µ¥¤¥º¤ÎÉ½¤ò»ØÄê¤·¤Þ¤¹¡£
É½¤Î½ñ¼°¤Ï¡¢ºÇ¾®¤«¤é½ç¤ËºÇÂç¤Þ¤Ç¤Î¡¢16 ¥Ó¥Ã¥ÈÉä¹æÌµ¤·À°¿ô¤Î¥ê¥¹¥È¤Ç¤¹¡£ ºÇ¾® MTU ¤Ï 68 ¤è¤ê¾®¤µ¤¯¤Æ¤Ï¤Ê¤ê¤Þ¤»¤ó¡£

option perform-mask-discovery *flag*;

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬ ICMP ¤ò»ÈÍÑ¤·¤Æ¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¥¯ÃµºÝ¤ò ¼Â»Ü¤¹¤Ù¤-¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ false ¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥Þ¥¹¥¯ÃµºÝ¤ò¼Â»Ü¤¹¤Ù¤-¤Ç¤Ê¤¤¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£ ÃÍ true ¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥Þ¥¹¥¯Ãµº÷¤ò¼Â»Ü¤¹¤Ù¤-¤Ç¤¢¤ë¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option policy-filter *ip-address ip-address*
[**,** *ip-address ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢È¥í¡¼¥«¥ë»ØÄê·ÐÏ©À©¸æ¤ËÂÐ¤¹¤ë¥Ý¥ê¥·¥Õ¥£¥ë¥¿¤ò»ØÄê¤·¤Þ¤¹¡£
¥Õ¥£¥ë¥¿¤Ï¡¢IP ¥¢¥É¥ì¥¹¤È¥Þ¥¹¥¯¤ÎÁÈ¤Î¥ê¥¹¥È¤«¤é¤Ê¤ê¡¢
ÅþÃå¤¹¤ë»ØÄê·ÐÏ©À©¸æ¤µ¤ì¤¿¥Ç¡¼¿¥°¥é¥àÍÑ¤Î¥Õ¥£¥ë¥¿¤È¤Ê¤ë
°¸Àè/¥Þ¥¹¥¯¤ÎÁÈ¤ò»ØÄê¤·¤Þ¤¹¡£

¼¡¥Û¥Ã¥×¥¢¥É¥ì¥¹¤¬¥Õ¥£¥ë¥¿¤Î¤¤¤º¤¤Ë¤âÅ¬¹ç¤·¤Ê¤¤»ØÄê·ÐÏ©À©¸æ¤µ¤ì¤¿
¥Ç¡¼¿¥°¥é¥à¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥ÇË´Þ¤¹¤Ù¤¤Ç¤¹¡£

¤µ¤é¤Ê¤ë¾ðÊó¤Ï STD 3 (RFC1122) ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£

option pop-server *ip-address* [**,** *ip-address...*]**;**

POP3 ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¤ÍøÍÑ²ÄÇ½¤Ê POP3 ¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£
¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option resource-location-servers *ip-address*
[**,** *ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¤ÍøÍÑ²ÄÇ½¤Ê RFC 887 ¥ê¥½¡¼¥¹¥í¥±¡¼¥·¥ç¥ó¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£
¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option root-path *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Î¥ë¡¼¥È¥Ç¥£¥¹¯¤¬´Þ¤Þ¤ì¤ë¥Ñ¥¹Ì¾¤ò»ØÄê¤·¤Þ¤¹¡£
¥Ñ¥¹¤Î½ñ¼°¤Ï¡¢NVT ASCII Ê¸»ú½¸¹ç¤ÎÊ¸»ú¤«¤é¤Ê¤ëÊ¸»úÎó¤Ç¤¹¡£

option router-discovery *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢RFC 1256 ¤ÇÄêµÁ¤µ¤ì¤ë¥ë¡¼¥ÃµºÝ (Router Discovery) µ¡¹¦¤ò »ÈÍÑ¤·¤Æ¡¢¥ë¡¼¥¿¤òÍ×ÀÁ¤¹Ù¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ false ¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥ë¡¼¥ÃµºÝ¤ò¼Â»Ü¤¹¤Ù¤Ç¤Ê¤¤¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£ ÃÍ true ¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Ï¥ë¡¼¥ÃµºÝ¤ò¼Â»Ü¤¹¤Ù¤Ç¤¢¤ë¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option router-solicitation-address *ip-address***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Î¥ë¡¼¥Í×ÀÁ¤ÎÁ÷½ÐÀè¥¢¥É¥ì¥¹¤ò»ØÄê¤·¤Þ¤¹¡£

option routers *ip-address* [**,** *ip-address...*]**;**

routers ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Î¥µ¥Ö¥Í¥Ã¥È¾å¤Ë¤¢¤ë¥ë¡¼¥¿¤Î IP ¥¢¥É¥ì¥¹¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥ë¡¼¥¿¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option slp-directory-agent *boolean ip-address* [**,** *ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢2 ¤Ä¤Î¹àÌÜ¤ò»ØÄê¤·¤Þ¤¹: 1 ¤Ä°Ê¾å¤Î¥µ¡¼¥Ó¥¹¥í¥±¡¼¥·¥ç¥ó¥×¥í¥È¥³¥ë¤Ç¥£¥ì¯¥È¥ê¨¡¼¥¸¥§¥ó¥È (Service Location Protocol Directory Agent) ¤Î IP ¥¢¥É¥ì¥¹¤È¡¢ ¤³¤ì¤é¤Î¥¢¥É¥ì¥¹¤Î»ÈÍÑ¤¬¶¯À©Åª¤«¤É¤¦¤«¤Ç¤¹¡£ ºÇ½é¤Î¥Ö¡¼¥ëÃÍ¤¬ true ¤Ç¤¢¤ì¤Ð¡¢SLP ¥¨¡¼¥¸¥§¥ó¥È¡¢¤¿¤ÀÍ¿¨¤é¤ì¤¿ IP ¥¢¥É¥ì¥¹¤Î¤ß¤ò»ÈÍÑ¤¹¤Ù¤¤Ç¤¹¡£ ÃÍ¤¬ false ¤Ç¤¢¤ì¤Ð¡¢SLP ¥¨¡¼¥¸¥§¥ó¥È¤Ï¡¢SLP ¥¨¡¼¥¸¥§¥ó¥È¤Î Ç½Æ°Åª¤â¤·¤õÆ°Åª¤Ê¥Þ¥ë¥Á¥-¥ã¥¹¥ÈÃµºÝ¤òÄÉ²Ã¤Ç¹Ô¤Ã¤Æ¤â½¤¤Þ¤ó (¾Ü¤·¤¯¤Ï RFC2165 ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤)¡£

ËÜ¥ª¥×¥·¥ç¥ó¤È slp-service-scope ¥ª¥×¥·¥ç¥ó¤Ë¤ª¤¤¤Æ¡¢ ¡ÖSLP ¥¨¡¼¥¸¥§¥ó¥È¡×¤È¤Ï¡¢DHCP ¥×¥í¥È¥³¥ë¤òÍÑ¤¤¤ÆÀßÄê¤µ¤ì¤¿¥Þ¥·¥ó¾å¤Ç
Æ°°î¤·¤Æ¤¤¤ë¥µ¡¼¥Ó¥¹¥í¥±¡¼¥·¥ç¥ó¥×¥í¥È¥³¥ë¥¨¡¼¥¸¥§¥ó¥È¤ò»Ø¤·¤Æ¤¤¤ë¤³¤È¤Ë
Ãí°Õ¤·¤Æ¤¯¤À¤µ¤¤¡£

¤Þ¤¿¡¢¤¤¤¯¤Ä¤«¤Î´ë¶È¤Ï SLP ¤ò NDS ¤È¸Æ¤ó¤Ç¤¤¤ë¤³¤È¤âµ¤¤òÉÕ¤±¤Æ¤¯¤À¤µ¤¤¡£ ¤â¤· NDS ¥Ç¥£¥ì¥¯¥È¥ê¥¨¡¼¥¸¥§¥ó¥È¤¬¤¢¤ê¡¢¤½¤Î¥¢¥É¥ì¥¹¤òÀßÄê¤¹¤ë¤É¬Í×¤¬ ¤¢¤ë¾ì¹ç¤Ï¡¢ slp-directory-agent ¥ª¥×¥·¥ç¥ó¤¬ÍøÍÑ¤Ç¤¤ë¤Ï¤ºÇ¤¹¡£

option slp-service-scope *boolean text*;

¥µ¡¼¥Ó¥¹¥í¥±¡¼¥·¥ç¥ó¥×¥í¥È¥³¥ë¤Î¥µ¡¼¥Ó¥¹¥¹¥³¡¼¥×¥ª¥×¥·¥ç¥ó¤Ï¡¢ 2 ¤Ä¤Î¹àÌÜ¤ò»ØÄê¤·¤Þ¤¹:
SLP ÍÑ¤Î¥µ¡¼¥Ó¥¹¥³¡¼¥×¤Î¥ê¥¹¥È¤È¡¢¤³¤Î¥ê¥¹¥È¤Î»ÈÍÑ¤¬¶¯À©Åª¤«¤É¦¤«¤Ç¤¹¡£
ºÇ½é¤Î¥Ö¡¼¥ëÃÍ¤¬ true ¤Ç¤¢¤ì¤Ð¡¢SLP ¥¨¡¼¥¸¥§¥ó¥È¤Ï¡¢ËÜ¥ª¥×¥·¥ç¥ó¤Ë¤è¤ê Äó¶¡¤µ¤ì¤¤¤ë¥¹¥³¡¼¥×¤Î¥ê¥¹¥È¤Î¤ß¤ò»ÈÍÑ¤¹¤Ù¤Ç¤¹¡£
¤½¦¤Ç¤Ê¤±¤ì¤Ð¡¢¤³¤Î¥ª¥×¥·¥ç¥ó¤ÇÄó¶¡¤µ¤ì¤ë¥ê¥¹¥È¤Ë¤Àè¤·¤Æ¡¢ ¤½¤ì¤¾¤ì¤Î¸ÇÍ-Åª¤ÎÀßÄê¤ò»È¤Ã¤Æ¤â¹½¤¤Þ¤»¤ó¡£

text Ê¸»úÎó¤Ï¡¢SLP ¥¨¡¼¥¸¥§¥È¤¬»ÈÍÑ¤¹¤Ù¤¥¹³¡¼¥×¤Î¡¢¥³¥ó¥Þ¶èÀÚ¤ê¤Î ¥ê¥¹¥È¤È¤·¤Æ¤¯¤À¤µ¤¤¡£ ¤³¤ì¾ÊÎ¬²ÄÇ½Ç¡¢¤½¤Î¾ì¹ç SLP ¥¨¡¼¥¸¥§¥È¤Ï¡¢¼«Ê¬¬ÃÎ¤Ã¤Æ¤¤¤ë
¤¹¤Ù¤Æ¤Î¥Ç¥£¥ì¥¯¥È¥ê¥¨¡¼¥¸¥§¥È¤Î¥³¡¼¥×¤Î°ì³ç¥ê¥¹¥È¤ò»È¤¤¤Þ¤¹¡£

option smtp-server *ip-address* [**,** *ip-address...*]**;**

SMTP ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê SMTP ¥µ¡¼¥Ð¤Î¥ê¥¹¥È¤ò »ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option static-routes *ip-address ip-address*
[**,** *ip-address ip-address...*]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬·ÐÏ©¥¥ã¥Ã¥·¥å¤ËÁÈ¤ß¹þ¤à¤Ù¤
ÀÅÅª·ÐÏ©¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ Æ±¤¸¸°¸Àè¤ËÂÐ¤·¤ÆÊ£¿ô¤Î·ÐÏ©¤¬»ØÄê¤µ¤ì¤Æ¤¤¤ë¾ì¹ç¤Ï¡¢
Í¥ÀèÅÙ¤¬Ä㤬¯¤Ê¤ë½ç½ø¤Ç¥ê¥¹¥È¤µ¤ì¤Þ¤¹¡£

·ÐÏ©¤Ï IP ¥¢¥É¥¹¥¤ÎÁÈ¤Î¥ê¥¹¥È¤«¤é¤Ê¤ê¤Þ¤¹¡£ ºÇ½é¤Î¥¢¥É¥¹¥¤Ï¸¸Àè¥¢¥É¥¹¥¤Ç¤¢¤ê¡¢ 2 ÈÖÌÜ¤Î¥¢¥É¥¹¥¤Ï¤½¤Î¸¸Àè¤ËÂÐ¤¹¤ë¥¿¤Î¥¢¥É¥¹¥¤Ç¤¹¡£

¥Ç¥Õ¥©¥ë¥È·ÐÏ© (0.0.0.0) ¤Ï¡¢ÀÅÅª·ÐÏ©¤ËÂÐ¤·¤Æ¤ÏÉÔÀµ¤Ê¸¸Àè¤Ç¤¹¡£
¥Ç¥Õ¥©¥ë¥È·ÐÏ©¤ò»ØÄê¤¹¤ë¤Ë¤Ï¡¢ **routers** ¥ª¥×¥·¥ç¥ó¤ò»ÈÍÑ¤·¤Æ¤¯¤À¤µ¤¤¡£
¤Þ¤¿¡¢ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥ì¥¹¤Ê IP ·ÐÏ©À©¸æ¤ò°Õ¿Þ¤·¤¿¤â¤Î¤Ç¤Ï ¤Ê¤¤¤³¤È¤ËÃí°Õ¤·¤Æ²¼¤µ¤¤¡£ ¤³¤ì¤Ï¥µ¥Ö¥Í¥Ã¥È¥Þ¥¯¯¤ò´Þ¤ó¤Ç¤¤¤Þ¤»¤ó¡£ ½ºß¡¢¥¯¥é¥¤¥ì¥¹¤Ê IP ·ÐÏ©À©¸æÏ¡¢¤â¤Ã¤È¤â¯Å¸³«¤µ¤ì¤Æ¤¤¤ë ·ÐÏ©À©¸æÉ¸½à¤Ê¤Î¤Ç¡ËÜ¥ª¥×¥·¥ç¥ó¤Ï¼ÂÅª¤ËÌµ°ÕÌ£¤Ç¤¹¡£ ¤½¤·¤Æ¡¢¥Þ¤¥¯¥í½Õ¥È DHCP ¯¥é¥¤¥¢¥ó¥È¤ò¤Ï¸¸á¤È¤¹¤ë¤è¯ÃÎ¤é¤ì¤¿ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ë¤Ï¼ÂÁõ¤µ¤ì¤Æ¤¤¤Þ¤»¤ó¡£

option streettalk-directory-assistance-server *ip-address*
[**,** *ip-address...*]**;**

StreetTalk Directory Assistance (STDA) ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê STDA ¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option streettalk-server *ip-address* [**,** *ip-address...*]**;**

StreetTalk ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê StreetTalk ¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Àè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option subnet-mask *ip-address***;**

¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¯¥ª¥×¥·¥ç¥ó¤Ï¡¢RFC 950 ¤Ë½¾¤Ã¤Æ¡¢¥¯¥é¥¤¥¢¥ó¥È¤Î¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¯¤ò»ØÄê¤·¤Þ¤¹¡£
¥¹³¡¼¥×Ãæ¤Î¤É¤³¤Ë¤â¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¯¥ª¥×¥·¥ç¥ó¤¬»ØÄê¤µ¤ì¤Æ¤¤¤Ê¤¤¾ì¹ç¡¢
ºÇ½ª¼êÃÊ¤È¤·¤Æ¡¢dhcpd ¤Ï¡¢¢¥¢¥É¥ì¥¹¤ò³ä¤êÅö¤Æ¤è¤¦¤È¤·¤Æ¤¤¤ë

¥Í¥Ã¥È¥ï¡¼¥¯¤Î¥µ¥Ö¥Í¥Ã¥È¤Àë¸À¤«¤é¡¢¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¥¯¤ò»ÈÍÑ¤·¤Þ¤¹¡£
¤·¤«¤·¡¢¥¢¥É¥ì¥¹¤ò³ä¤êÅö¤Æ¤è¤¦¤È¤·¤Æ¤¤¤ë¥Í¥Ã¥È¥ï¡¼¥¯¤Î¥¹³¡¼¥×Ãæ¤Î ¤É¤Î¤è¤¦¤Ê ¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¥¯¥ª¥×¥·¥ç¥óÀë¸À¤Ç¤¢Ã¤Æ¤â¡¢ ¥µ¥Ö¥Í¥Ã¥È¤Àë¸À¤Ç»ØÄê¤µ¤ì¤¿¥µ¥Ö¥Í¥Ã¥È¥Þ¥¹¥¯¤ËÍ¥Àè¤·¤Þ¤¹¡£

option subnet-selection *string***;**

(Í×µá¤¬½Ð¤µ¤ì¤¿¥µ¥Ö¥Í¥Ã¥È¤Ë¤Ä¤Ê¤¬¤ì¤¿¥ê¥ì¡¼¥µ¡¼¥Ð¤Î¥¢¥É¥ì¥¹¤Ë´ð¤Å¤¤¤Æ) Ä쾄ïÁªÂò¤µ¤ì¤ë¤Ç¤¢¤í¦¤â¤Î¤Ç¤Ï¤Ê¤¤¥µ¥Ö¥Í¥Ã¥È¤Î¥¢¥É¥ì¥¹¤¬É¤Í×¤Ê¾ì¹ç¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤¬Á÷½Ð¤·¤Þ¤¹¡£ RFC3011 ¤ò»²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£ ¤³¤Î¥µ¡¼¥Ð¤Ë¤ª¤¤¤Æ»È¤ï¤ì¤ë¥ª¥×¥·¥ç¥óÊ¥ó¥Ð¤Ï 118 ¤Ç¤¹¡£ ¤³¤Î¥Ê¥ó¥Ð¤Ï°ÊÁ°¤«¤é¤º¤Ã¤ÈÄêµÁ¤µ¤ì¤Æ¤¤¤¿¥Ê¥ó¥Ð¤Ç¤Ï¤Ê¤¯¡¢ °ã¤¦ÃÍ¤ò»ÈÍÑ¤¹¤ë¥¯¥é¥¤¥¢¥ó¥È¤âÂ¸ºß¤¹¤ë¤³¤È¤ËÃí°Õ¤·¤Æ¤¯¤À¤µ¤¤¡£ ¤³¤Î¥ª¥×¥·¥ç¥ó¤Î»ÈÍÑ¤Ï¾¯¡¹¼Â¸³Å¤ª¤Ç¤¢¤ë¤È¹Í¤¨¤ë¤Ù¤¤Ç¤·¤ç¤¦!

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥µ¡¼¥Ð¤Ç¤Ï¥æ¡¼¥¶¤¬ÀßÄê¤¹¤ë¤³¤È¤Ï¤Ç¤¤Þ¤»¤ó¡£

option swap-server *ip-address***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Î¥¹¥ï¥Ã¥×¥µ¡¼¥Ð¤Î IP ¥¢¥É¥ì¥¹¤ò»ØÄê¤·¤Þ¤¹¡£

option tcp-keepalive-garbage *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢Å¤¤¼ÂÁõ¤È¤¸ß´¹À¤Î¤¿¤á¤Ë¡¢´¥ß¤Î¥ª¥¯¥Æ¥Ã¥È¤È°ì½ï¤Ë¡¢ TCP ¥-¡¼¥×¥¢¥é¥¤¥Ö¥á¥Ã¥»¡¼¥¸¸ò¯¥é¥¤¥¢¥ó¥È¤¬Á÷¤ë¤Ù¤¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ false ¤Ï¡¢´¥ß¤Î¥ª¥¯¥Æ¥Ã¥È¤òÁ÷¤ë¤Ù¤¤Ç¤Ê¤¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£ ÃÍ true ¤Ï¡¢´¥ß¤Î¥ª¥¯¥Æ¥Ã¥È¤òÁ÷¤ë¤Ù¤¤Ç¤¢¤ë¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option tcp-keepalive-interval *uint32***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Î TCP ¤¬¥¡¼¥×¥¢¥é¥¤¥Ö (keepalive) ¥á¥Ã¥»¡¼¥¸¸ò TCP ÀÜÂ³¾å¤ËÁ÷¿®¤¹¤ëÁ°¤ËÂÔ¤Ä¤Ù¤´Ö³Ö (ÉÃÃ±°Ì) ¤ò»ØÄê¤·¤Þ¤¹¡£ »þ´Ö¤Ï 32 ¥Ó¥Ã¥ÈÉä¹æÌµ¤·À°¿ô¤Ç»ØÄê¤·¤Þ¤¹¡£ ÃÍ 0 ¤Ï¡¢¥¢¥×¥ê¥±¡¼¥·¥ç¥ó¤¬ÌÀ¼¨Åª¤ËÍ×µá¤·¤Ê¤¸¸Â¤ê¡¢¯¥é¥¤¥¢¥ó¥È¤¬ ÀÜÂ³¾å¤Ë¥-¡¼¥×¥¢¥é¥¤¥Ö¥á¥Ã¥»¡¼¥¸¸òÀ¸À®¤¹¤Ù¤¤Ç¤Ê¤¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option tftp-server-name *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï TFTP ¥µ¡¼¥Ð¤ò»ØÄê¤¹¤ë¤Î¤Ë»ÈÍÑ¤µ¤ì¡¢¯¥é¥¤¥¢¥ó¥È¤¬ ¥µ¥Ý¡¼¥È¤·¤Æ¤¤¤ë¾ì¹ç¤Ë¤Ï **server-name** Àë¸À¤ÈÆ±¤¸¸ú²Ì¤ò»ý¤Á¤Þ¤¹¡£ BOOTP ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢ËÜ¥ª¥×¥·¥ç¥ó¤ò¥µ¥Ý¡¼¥È¤·¤Ê¤¤¤Ç¤·¤ç¤¦¡£ DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ë¤è¤Ã¤Æ¤Ï¥µ¥Ý¡¼¥È¤·¤Æ¤¤¤¤â¤Î¤¬¤¢¤ê¡¢ ¼Âº¶Ý¤¬¿Ü¤È¤·¤Æ¤¤¤ë¤â¤Î¤¬¤¢¤ê¤Þ¤¹¡£

option time-offset *int32***;**

time-offset ¥ª¥×¥·¥ç¥ó¤Ï¡¢¶¨ÄêÀ¤³¦»þ (UTC) ¤ò´ðÅÀ¤È¤·¤Æ¡¢ ¥¯¥é¥¤¥¢¥ó¥È¤Î¥µ¥Ö¥Í¥Ã¥È¤Î¥ª¥Õ¥»¥Ã¥È¤òÉÃ¤Ç»ØÄê¤·¤Þ¤¹¡£

option time-servers *ip-address* [, *ip-address*...]**;**

time-server ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½Ê RFC 868 »þ¹ï¥µ¡¼¥Ð¤Î ¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥ÀèºµÌ¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¯¤À¤µ¤¤¡£

option trailer-encapsulation *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢ARP ¥×¥í¥È¥³¥ë»È¤ÍÑ»þ¤Ë¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥È¥ì¥¤¥é»È¤ÍÑ¤Î ¥Í¥´¥·¥¨¡¼¥·¥ç¥ó (RFC 893 [14]) ¤ò¤¹¤Ù¤¤«¤ò»ØÄê¤·¤Þ¤¹¡£ ÃÍ false ¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥È¥ì¥¤¥é»È¤ÍÑ¤ò»î¤ß¤ë¤Ù¤¤Ç¤Ê¤¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£ ÃÍ true ¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬¥È¥ì¥¤¥é»È¤ÍÑ¤ò»î¤ß¤ë¤Ù¤¤Ç¤¢¤ë¤È°ÕÌ£¤·¤Þ¤¹¡£

option uap-servers *text***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶Ç§¾Ú¥×¥í¥È¥³¥ë (UAP) ¤ËÊñ¤Þ¤ì¤¿Ç§¾ÚÍ×µá¤ò ½èÍý¤¹¤ëÇ½ÎÏ¤Î¢¤ë¥æ¡¼¥¶Ç§¾Ú¥µ¡¼¥Ó¥¹¤ò¤½¤ì¤¾¤ì»Ø·¤Æ¤¤¤ë URL

¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ UAP ¥µ¡¼¥Ð¤Ï HTTP 1.1 ÀÜÂ³¤â SSLv3 ÀÜÂ³¤â¼õ¤±¼è¤ë³¤È¤¬¤Ç¤-¤Þ¤¹¡£ ¥ê¥¹¥È¤Ë´Þ¤Þ¤ì¤¿ URL ¤Ë¥Ý¡¼¥ÈÉôÊ¬¤¬´Þ¤Þ¤ì¤Æ¤Ê¤¤¾ì¹ç¤Ï¡¢ ÄÌ¾ï¤Î¥Ç¥Õ¥©¥ë¥È¥Ý¡¼¥È¤¬²¾Äê¤µ¤ì¤Þ¤¹ (¤Ä¤Þ¤ê http ¤Ë¤Ï 80 ÈÖ¡¢https ¤Ë¤Ï 443 ÈÖ)¡£ ¥ê¥¹¥È¤Ë´Þ¤Þ¤ì¤¿ URL ¤Ë¥Ñ¥¹¤ÎÉôÊ¬¤¬´Þ¤Þ¤ì¤Æ¤Ê¤¤¾ì¹ç¤Ï¡¢ ¥Ñ¥¹¤Ï /uap ¤È²¾Äê¤µ¤ì¤Þ¤¹¡£ 2 ¤Ä°Ê¾å¤Î URL ¤¬¤³¤Î¥ê¥¹¥È¤Ë»ØÄê¤µ¤ì¿¾ì¹ç¡¢URL ¤Ï¶õÇò¤Ç¶èÀÚ¤é¤ì¤Þ¤¹¡£

option user-class *string***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢æ¡¼¥¶¤¬¼±ÊÌ¾ðÊó¤ò¯¯é¥¤¥¢¥ó¥È¤Ë»ØÄê¤¹¤ë¼êÃÊ¤È¤·¤Æ¡¢ ¤¤¤¯¤Ä¤«¤Î DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ç»È¤ï¤ì¤Þ¤¹¡£ ¤³¤ì¤Ï vendor-class-identifier ¥ª¥×¥·¥ç¥ó¤È¤Æ±ÍÍ¤Ë»È¤ï¤ì¤Þ¤¹¤¬¡¢ ¤½¤ÎÃÍ¤Ï¡¢¥Ù¥ó¥À¤Ç¤Ï¤Ê¤¯¡¢¥æ¡¼¥¶¤Ë¤è¤Ã¤Æ»ØÄê¤µ¤ì¤Þ¤¹¡£ ºÇ¶á¤Î¤Û¤È¤ó¤É¤Î DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢¤³¤Î¼±ÊÌ»Ò¤ËÃÍ¤ò»ØÄê¤¹¤ë¿¿á¤Î ¥æ¡¼¥¶¥¤¥ó¿¥Õ¥§¡¼¥¹¤òÈ÷¤¨¤Æ¤¤¤Þ¤¹¡£ ¤³¤Î¼±ÊÌ»Ò¤Ï¡¢ÄÌ¾ï¥Æ¥¹¥ÈÊ¸»úÎó¤Ç¤¹¡£

option vendor-class-identifier *string***;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥Ù¥ó¥À¿¥¤¥×¤ä¡¢²ÄÇ½¤Ç¤¢¤ì¤Ð DHCP ¥¯¥é¥¤¥¢¥ó¥È¤ÎÀßÄê¤ò ¼±ÊÌ¤¹¤ë¿¿á¤Ë¡¢¤¤¤¯¤Ä¤«¤Î DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ç»È¤ï¤ì¤Þ¤¹¡£ ¤³¤Î¾ðÊó¤ÎÆâÍÆ¤Ï¡¢¥Ù¥ó¥À¸ÇÍ¤Î¥Ð¥¤¥ÈÊ¸»úÎó¤Ç¡¢½à¤Ç¤Ïµ¬Äê¤µ¤ì¤Æ¤¤¤Þ»¤ó¡£ ¥¯¥é¥¤¥¢¥ó¥È¤¬Á÷½Ð¤¹¤ë¥Ù¥ó¥À¥¯¥é¥¤¥¢¥ó¥È¤ò³ÎÇ§¤¹¤ë¤Ë¤Ï¡¢ °Ê²¼¤ÎÀßÄê¤ò DHCP ¥µ¡¼¥ÐÀßÄê¥Õ¥¡¥¤¤ë¤Ë²Ã¤¨¤Æ¤¯¤À¤µ¤¤:

set vendor-class option vendor-class-identifier;

¤³¤ÎÀßÄê¤Ï¡¢DHCP ¥µ¡¼¥Ð¤Î¥ê¡¼¥Ç¡¼¥Ù¡¼¥¹¥Õ¥¡¥¤¤ëÃæ¤Î¡¢ °Ê²¼¤Î¤è¦¤Ê set Ê¸¤ò»ý¤Ä vendor-class-identifier ¥ª¥×¥·¥ç¥ó¤ò Á÷¤Ã¤Æ¤¯¤ë¥¯¥é¥¤¥¢¥ó¥È¤¹¤Ù¤Æ¤Î¨¥ó¥È¥ê¤Ëºî¤·¤Þ¤¹¡£

set vendor-class "SUNW.Ultra-5_10";

vendor-class-identifier ¥ª¥×¥·¥ç¥ó¤Ï¡¢ÄÌ¾ï DHCP Server ¤Ë¤è¤Ã¤Æ¡¢ **vendor-encapsulated-options** ¥ª¥×¥·¥ç¥óÃæ¤ÇÊÖ¤µ¤ì¤ë¥ª¥×¥·¥ç¥ó¤ò·èÄê¤¹¤ë¤Î¤è¦¤Ë¡¢¤µ¤é¤Ê¤¾ðÊó¤Ï¡¢dhcpd.conf ¥Þ¥Ë¥å¥¢¥ë¥Ú¡¼¥¸¤Î VENDOR ENCAPSULATED OPTIONS ¤Î ¾Ï¤ò»²¾È¤·¤Æ¯¤À¤µ¤¤¡£

option vendor-encapsulated-options *string***;**

vendor-encapsulated-options ¥ª¥×¥·¥ç¥ó¤Ï¡¢1 ¤Ä¤Î¥Ù¥ó¥À¸ÇÍÃÍ¡¢ ¤â¤·¤¯¯¤Ï 1 ¤Ä¤Þ¤¿¤Ï¤½¤ì°Ê¾å¤Î¥Ù¥ó¥À¸ÇÍ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤´Þ¤ß¤Þ¤¹¡£ ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢ÄÌ¾ï DHCP ¥µ¡¼¥ÐÀßÄê¥Õ¥¡¥¤¤ë¤ÇÀßÄê¤µ¤ì¤ë¤â¤Î¤Ç¤Ï ¤¢¤ê¤Þ¤»¤ó¡£ ÄÌ¾ï¤Ï¡¢¥Ù¥ó¥À¥¯¥é¥¤¥Ù¥ó¥ÀèÄËÄêµÁ¤µ¤ì¡¢ ¥Ù¥ó¥À¥¯¥é¥¤¥µ¥Ö¥ª¥×¥·¥ç¥ó¤¬Äêµ¤¤ì¡¢¤½¤ÎÃÍ¤¬ ÄêµÁ¤µ¤ì¡¢DHCP ¥µ¡¼¥Ð¤Ï¤½¤ì¤é¤ò¤â¤È¤Ë±Åú¤òÁÈ¤ß¾å²¤Þ¤¹¡£

¤è¤¯ÃÎ¤é¤ì¤¿ DHCP ¥¯¥é¥¤¥¢¥ó¥È¥Ù¥ó¥À (º£¤Î¤È³¤í Microsoft Windows 2000 DHCP ¥¯¥é¥¤¥¢¥ó¥È) ¸þ¤±¤Î¤¤¯¤Ä¤«¤Î¥Ç¥Õ¥©¥ë¥È¤ÎÆ°ºî¤Ç¤Ï¡¢ ¤³¤Î¥ª¥×¥·¥ç¥ó¤Ï¼«Æ°Å²¤ËÀßÄê¤µ¤ì¤Þ¤¹¤¬¡¢¤½¤ÎÂ¾¤Î¤â¤Î¤Ë´Ø¤·¤Æ¤Ï¡¢ ¼êÆ°¤ÇÀßÄê¤·¤Ê¤±¤ì¤Ð¤Ê¤ê¤Þ¤»¤ó¡£ ¾Ü¤Ù¤Ï *dhcpd.conf ¤Î VENDOR ENCAPSULATED OPTIONS ¤Î¾Ï¤ò »²¾È¤·¤Æ¤¯¤À¤µ¤¤¡£*

option www-server *ip-address* [**,** *ip-address*...]**;**

WWW ¥µ¡¼¥Ð¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê WWW ¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥µ¡¼¥Ð¤Ï¡¢Í¥Èè¤µ¤ì¤¤¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

option x-display-manager *ip-address* [**,** *ip-address*...]**;**

ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤¬ÍøÍÑ²ÄÇ½¤Ê X Window System ¥Ç¥£¥¹¥×¥ì¥¤¥Þ¥Í¡¼¥¸¥ã¤ò¼Â¹Ô¤·¤Æ¤¤¤ë¥·¥¹¥Æ¥à¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¥¢¥É¥ì¥¹¤Ï¡¢Í¥Èè¤µ¤ì¤ë¤â¤Î¤«¤é½ç¤Ë¥ê¥¹¥È¤·¤Æ¤¯¤À¤µ¤¤¡£

¥ê¥ì¡¼¯¡¼¸¥§¥ó¥È¾ðÊó¥ª¥×¥·¥ç¥ó

IETF ¥É¥é¥Õ¥È draft-ietf-dhc-agent-options-11.txt ¤Ë¤Ï¡¢ DHCP ¥ê¥ì¡¼¯¡¼¸¥§¥ó¥È¤¬ DHCP ¥Ñ¥±¥Ã¥È¤ò DHCP ¥µ¡¼¥ÐËå¾Á÷¤¹¤ëºÝ¡¢ DHCP ¥Ñ¥±¥Ã¥È¤ËÊÕ²Ã¤¹¤ë³¤È¤Î¤Ç¤-

¤ë°ìÏ¢¤Î¥«¥×¥»¥ë²½¤µ¤ì¤¿¥ª¥×¥·¥ç¥ó¤¬ ÄêµÁ¤µ¤ì¤Æ¤¤¤Þ¤¹¡£
¥µ¡¼¥Ð¤Ï¡¢¤³¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤Ë´ð¤Å¤¡¢¥¢¥É¥ì¥¹äÅö¤Î·èÄê (¤ä¡¢¤½¤ÎÂ¾¤ÎÈ½ÃÇ)
¤ò¹Ô¤¦¤³¤È¤¬Ç¤¤Þ¤¹¡£ ¤Þ¤¿¥µ¡¼¥Ð¤Ï¡¢¥ê¥ì¡¼¥¨¡¼¥¸¥§¥ó¥È¤òÄÌ¤·¤ÆÊÖ¤µ¤ì¤ë¤É¤Î¥Ñ¥±¥Ã¥È¤Ë¤â
¤³¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤òÆþ¤ì¤ÆÊÖ¤·¤Þ¤¹¡£
¤³¤ì¤Ë¤è¤Ã¤Æ¥ê¥ì¡¼¥¨¡¼¥¸¥§¥ó¥È¤Ï¡¢ÇÛÁ÷¤ä¥¢¤«¥¦¥ó¥Æ¥£¥ó¥°¤Ê¤É¤ò
¹Ô¤¦¤¿¤á¤Ë¡¢¤³¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤Ë´Þ¤Þ¤ì¤ë¾ðÊó¤òÍøÍÑ¤Ç¤¤Þ¤¹¡£

¸½ºß¤Î¥É¥é¥Õ¥È¤Ë¤Ï 2 ¤Ä¤Î¥ª¥×¥·¥ç¥ó¤¬ÄêµÁ¤µ¤ì¤Æ¤¤¤Þ¤¹¡£ DHCP
¥µ¡¼¥Ð¤Ç¤³¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤ò»²¾È¤¤ë¤Ë¡¢¥ª¥×¥·¥ç¥ó¶õ´ÖÌ¾ "agent"
¤Î¤¢¤È¤Ë¤Ô¥ê¥ª¥É¤ò¤Ä¤±¡¢¤½¤Î¸å¤Ë¥ª¥×¥·¥ç¥óÌ¾¤òÂ³¤±¤Æ¤¯¤À¤µ¤¤¡£
¥µ¡¼¥Ð¤Ç¤³¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤òÄêµÁ¤¤ë¤³¤È¤ò¡¢ ÄÌ¾ï¤¢¤Þ¤êÍ-
¸ú¤Ç¤Ï¤¢¤ê¤Þ¤»¤ó¤¬¡¢µöÍÆ¤µ¤ì¤Æ¤¤¤Þ¤¹¡£
¤³¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤Ç¤Ï¥µ¥Ý¡¼¥È¤µ¤ì¤Æ¤¤¤Þ¤»¤ó¡£

option agent.circuit-id *string***;**

circuit-id ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤Ï¡¢¯¥é¥¤¥¢¥ó¥È¤«¤é¥µ¡¼¥Ð¤Ø¤Î DHCP ¥Ñ¥±¥Ã¥È¤ò
¼õ¤±¼è¤Ã¤¿¥µ¡¼¥Ã¥È¤ò¼¨¤¤¡¢¥¨¡¼¥¸¥§¥ó¥È¥í¡¼¥«¥ë¤Ê¥µ¡¼¥¥Ã¥È¼±ÊÌ»Ò¤ò
¥¨¥ó¥³¡¼¥É¤·¤Æ¤¤¤Þ¤¹¡£ ¤³¤ì¤Ï¡¢DHCP ±þÅú¤òÅ¬ÀÚ¤Ê¥µ¡¼¥¥Ã¥È¤Ø¤ÈÁ÷¤êÊÖ¤»¤ë¤è¤¦¡¢
¥¨¡¼¥¸¥§¥ó¥È¤Ë¤è¤Ã¤Æ»È¤ï¤¤¤³¤È¤ò°Õ¿Þ¤·¤Æ¤¤¤Þ¤¹¡£
¸½ºß¡¢¤³¤Î¥ª¥×¥·¥ç¥ó¤Î½ñ¼°¤ÎÄêµÁ¤Ï¥Ù¥ó¥À°ÍÂ¸¤È¤Ê¤Ã¤Æ¤ª¤ê¡¢
Â¿Ê¬¤³¤Î¤Þ¤Þ¤Ä¤µ¤ì¤ë¤Ç¤·¤ç¤¦¡£ ¤·¤«¤·¾Íè³¤Î½ñ¼°¤¬É¸½à²½¤µ¤ì¤ë²ÄÇ½À-
¤â¡¢½ºß¤Î¥É¥é¥Õ¥È¤Ë¤Ï»Ä¤µ¤ì¤Æ¤¤¤Þ¤¹¡£

option agent.remote-id *string***;**

remote-id ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤Ï¡¢¤µ¡¼¥¥Ã¥È¤Î½ªÃ¼¤Î¥ê¥â¡¼¥È¥Û¥¹¥È¤Î
¾ðÊó¤ò¥¨¥ó¥³¡¼¥É¤·¤Æ¤¤¤Þ¤¹¡£ ¤³¤ì¤Ë´Þ¤Þ¤ì¤ë¤Ç¤¢¤í¡¦¾ðÊó¤Ï¡¢¤Î¤è¡¦¤Ê¤â¤Î¤Ç¤¹¡£ ¸Æ½Ð¸µ
ID ¾ðÊó¡¢¤æ¡¼¥¶Ì¾¾ðÊó¡¢¤ê¥â¡¼¥È ATM ¥¢¥É¥ì¥¹¡¢¤±¡¼¥Ö¥ë¥â¥Ç¥à ID¡¢
¤½¤ÎÂ¾¤ÎÆ±ÍÍ¤Ê¾ðÊó¡£ ¸¶Â§Å°¤Ë¤Ï¡¢¤³¤Î¥ª¥×¥·¥ç¥ó¤Î°ÕÌ£¤Ï¤Á¤ã¤ó¤ÈÄêµÁ¤µ¤ì¤Æ¤¤¤Þ¤»¤ó¡£
¤·¤«¤·ÄÌ¾ï¡¢¥µ¡¼¥¥Ã¥È¤ÎÆÃÄê²Î¥ê¥â¡¼¥È¨¥ó¥É¤ËÂÐ¤·¤Æ°ì°Õ¤Ç¤¢¤ë¤è¤¦
´ÉÍý¾åÊÝ¾Ú¤µ¤ì¤¿¡¢¤Ê¤ó¤é¤«¤Î¥ª¥Ö¸¸§¯¥È¤È¹Í¨¤ë¤Ù¤¤â¤Î¤Ç¤¹¡£

¯¥é¥¤¥¢¥ó¥È FQDN ¥µ¥Ö¥ª¥×¥·¥ç¥ó

¸½ºß¡¢¤¥¤ó¥¿¡¼¥Í¥Ã¥È¥É¥é¥Õ¥È draft-ietf-dhc-fqdn-option-00.txt ¤Ç ÄêµÁ¤µ¤ì¤Æ¤¤¤ë¯¥é¥¤¥¢¥ó¥È
FQDN ¥ª¥×¥·¥ç¥ó¤Ï¡¢¤Þ¤ÀÉ¸½à¤È¤Ê¤Ã¤Æ¤Ï¤¤¤Þ¤»¤ó¡£ ¤·¤«¤·¤¹¤Ç¤Ë½½Ê¬¹-
¤¯ÍøÍÑ¤µ¤ì¤Æ¤ª¤ê¡¢²æ¹¤â¤³¤ì¤ò¼ÂÁÃõ¤·¤Æ¤¤¤Þ¤¹¡£
¥ª¥×¥·¥ç¥ó¤Î½ñ¼°¤¬Ê£»¨¤Ê¤¿¤á¡¢¤³¤³¤Ç¤Ï¡¢Ã±ÆÈ¤Î¥ª¥×¥·¥ç¥ó¤Ç¤Ï¤Ê¤¯¡¢
¥µ¥Ö¥ª¥×¥·¥ç¥ó¶õ´Ö¤Ë¼ÂÁÃõ¤·¤Æ¤¤¤Þ¤¹¡£
°ìÈÌÅª¤Ë¤Ï¡¢¤ËÜ¥ª¥×¥·¥ç¥ó¤Ï¡¢¥æ¡¼¥¶¤Ë¤Ï¤Ê¤¿¤Ã±ÆÈ¤Î¥ª¥×¥·¥ç¥ó¤Ç¤Ï¤Ê¤¯¡¢ ¼«Æ° DNS
¹¹¿·¥·¥¹¥Æ¥à¤Î°ìÉô¤È¤·¤Æ»È¤¤ë¤Ù¤¤â¤Î¤Ç¤¹¡£

option fqdn.no-client-update *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤¬¯¥é¥¤¥¢¥ó¥È¤«¤éÁ÷½Ð¤µ¤ì¤¿¾ì¹ç¡¢¤³¤ì¤ç true ¤Ç¤¢¤ì¤Ð¡¢¢
¯¥é¥¤¥¢¥ó¥È¤Ï¼«Ê¬¤Î A ¥ì¥³¡¼¥É¤ò¹¹¿·¤·¤Ê¤¤¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£
¥µ¡¼¥Ð¤«¤é¯¥é¥¤¥¢¥ó¥È¤ËÁ÷½Ð¤µ¤ì¤¿¾ì¹ç¤Ï¡¢¤Ê¬¤Î A ¥ì¥³¡¼¥É¤ò¹¹¿·¤¹¤ë
¤Ù¤¤Ç¤Ï¤Ê¤¤ ¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option fqdn.server-update *flag***;**

ËÜ¥ª¥×¥·¥ç¥ó¤¬¯¥é¥¤¥¢¥ó¥È¤«¤é¥µ¡¼¥Ð¤ËÁ÷½Ð¤µ¤ì¤¿¾ì¹ç¡¢ ¥µ¡¼¥Ð¤Ë¯¥é¥¤¥¢¥ó¥È¤Î A
¥ì¥³¡¼¥É¤Î¹¹¿·µá¤·¤Æ¤¤¤Þ¤¹¡£ ¥µ¡¼¥Ð¤«¤éÁ÷½Ð¤µ¤ì¤¿¾ì¹ç¡¢¥µ¡¼¥Ð¤¬¯¥é¥¤¥¢¥ó¥È¤Î A
¥ì¥³¡¼¥É¤ò ¹¹¿·¤·¤¿ (¤â¤·¯¤Ï¤â¦¤¹°¹¹¿·¤¹¤ë¤È³¤í) ¤Ç¤¢¤ë¤³¤È¤ò°ÕÌ£¤·¤Þ¤¹¡£

option fqdn.encoded *flag***;**

true ¤Ç¤¢¤Ã¤¿»þ¡¢¥ª¥×¥·¥ç¥ó¤Ë´Þ¤Þ¤ì¤ë¥É¥á¥¤¥óÌ¾¤¬¡¢ ¤¿¤À¤Î ASCII ¥Æ¥¥¹¥È¤Ç¤Ï¤Ê¤¯¡¢DNS
¥ï¥¤¥ä¥Õ¥©¡¼¥Þ¥Ã¥È¤Ç ¥¨¥ó¥³¡¼¥É¤µ¤ì¤Æ¤¤¤ë¤³¤È¤ò¼¨¤·¤Æ¤Þ¤¹¡£ ¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢¼«Ê¬¤¬
FQDN ¥ª¥×¥·¥ç¥ó¤Î DNS ¥ï¥¤¥ä¥Õ¥©¡¼¥Þ¥Ã¥È¤ò
¥µ¥Ý¡¼¥È¤·¤Æ¤Ê¤¤¾ì¹ç¡¢ÄÌ¾ï¤³¤Î¥ª¥×¥·¥ç¥ó¤ò false ¤ËÀßÄê¤·¤Þ¤¹¡£
¥µ¡¼¥Ð¤Ï¾ï¤Ë¡¢¯¥é¥¤¥¢¥ó¥È¤¬ÀßÄê¤·¤¿¤Î¤ÈÆ±¤¸ÃÍ¤òÀßÄê¤·¤ÆÊÖ¤¹¤Ù¤¤Ç¤¹¡£

¤³¤ÎÃÍ¤¬ÀßÄê¥Õ¥¡¥¤¥ë¤ËÀßÄê¤µ¤ì¤Æ¤¤¤¿»þ¤Ï¡¢*fqdn.fqdn* ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤ò ¥¨¥ó¥³¡¼¥É¤¹¤ë¥Õ¥©¡¼¥Þ¥Ã¥È¤òÀ©¸æ¤·¤Þ¤¹¡£

option fqdn.rcode1 *flag***;**

option fqdn.rcode2 *flag***;**

¤³¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤Ï¤½¤ì¤¾¤ì¡¢A ¥ì¥³¡¼¥É¤È PTR ¥ì¥³¡¼¥É¤Î¹¿··ë²Ì¤ò¼¨¤·¤Þ¤¹¡£ ¤³¤ì¤é¤Ï¡¢DHCP ¥µ¡¼¥Ð¤«¤é DHCP ¥¯¥é¥¤¥¢¥ó¥È¤Ø¤Î¤ßÁ÷¤é¤ì¤Þ¤¹¡£ ¤³¤ì¤é¤Î¥Õ¥£¡¼¥ë¥É¤ÎÃÍ¤Ï¡¢DNS ¥×¥í¥È¥³¥ëµ¬³Ê¤Ë¤è¤êÄêµÁ¤µ¤ì¤Æ¤¤¤Þ¤¹¡£

option fqdn.fqdn *text***;**

¥¯¥é¥¤¥¢¥ó¥È¤¬»ÈÍÑ¤òË¾¤à¥É¥á¥¤¥óÌ¾¤ò»ØÄê¤·¤Þ¤¹¡£
¤³¤ì¤Ï´°Á´½¤¾þ¤µ¤ì¤¿¥É¥á¥¤¥óÌ¾¤Ç¤â¡¢Ã±°ì¤Î¥é¥Ù¥ë¤Ç¤â¤ê½¤¤¤Þ¤»¤ó¡£ ¤â¤·Ì¾Á°¤Ë Ê¸»ú¤¬´Þ¤Þ¤ì¤Ê¤±¤ì¤Ð¡¢¤½¤ÎÌ¾Á°¤Ï´°Á´½¤¾þ¤µ¤ì¤Æ¤ª¤é¤º¡¢
¥µ¡¼¥Ð¤ÏÄÌ¾ï¡¢¥í¡¼¥«¥ë¤ËÄêµÁ¤µ¤ì¿¥É¥á¥¤¥óÃæ¤Î¤½¤ÎÌ¾Á°¤ò¹¹¿·¤·¤Þ¤¹¡£

¤â¤·¤³¤ì¤é¤Î¥µ¥Ö¥ª¥×¥·¥ç¥ó¤ò»ÈÍÑ¤·¤è¡¦¤È»×¤Ã¤Æ¤¤¤ë¤Î¤Ç¤¢¤ì¤Ð¡¢ ¥¯¥é¥¤¥¢¥ó¥È FQDN ¥ª¥×¥·¥ç¥ó¤Î¥É¥é¥Õ¥È (¤â¤·¤¯¤Ï¡¢É¸½à¤Ë¤Ê¤Ã¤¿»þ¤Ï¤½¤ÎÉ¸½à) ¤ò»²¾È¤¹¤ë¤³¤È¤ò¶¯¤¯¿ä¾©¤·¤Þ¤¹¡£ ¤³¤ÎÊ¸½ñ¤Ï¡¢¤¹¤Î¥É¥é¥Õ¥È¤ËÈæ¤Ù¤ÆÂç»¨ÇÄ¤ÇÉÔ´°Á´¤Ç¤¢¤ê¡¢ ¥¯¥é¥¤¥¢¥ó¥È FQDN ¥ª¥×¥·¥ç¥óµ¬³Ê¤ò¤¹¤Ç¤ËÍý²ò¤·¤Æ¤¤¤ë¿Í¤Ë»²¾È¤µ¤ì¤ë¤³¤È¤ò Ã±¤Ë°Õ¿Þ¤·¤Æ¤¤¤ë¤â¤Î¤Ç¤¹¡£

NetWare/IP ¥µ¥Ö¥ª¥×¥·¥ç¥ó

RFC2242 ¤Ï¡¢Novell ¤Î NetWare/IP ¥¯¥é¥¤¥¢¥ó¥ÈÍÑ¤Î¥«¥×¥»¥ë²½¤µ¤ì¤¿ ¥ª¥×¥·¥ç¥ó¤ÎÁÈ¤òÄêµÁ¤·¤Æ¤¤¤Þ¤¹¡£ DHCP ¥µ¡¼¥Ð¤Ëª¤¤¤Æ¤³¤¤é¤Ï¥ª¥×¥·¥ç¥ó¤ò»ÈÍÑ¤¹¤ë¤Ï¡¢¥ª¥×¥·¥ç¥ó¶õ´ÖÌ¾ "nwip" ¤Î¸å¤Ë¥Ô¥ê¥ª¥É¤ò¤Ä¤±¡¢¤½¤Î¸å¤Ë¥ª¥×¥·¥ç¥óÌ¾¤òÂ³¤±¤Æ¤¯¤À¤µ¤¤¡£ °Ê²¼¤Î¥ª¥×¥·¥ç¥ó¤¬»ØÄê¤Ç¤-¤Þ¤¹:

option nwip.nsq-broadcast *flag***;**

true ¤Ç¤¢¤Ã¤¿¾ì¹ç¡¢¥¯¥é¥¤¥¢¥ó¥È¤Ï¡¢NetWare/IP ¥µ¡¼¥Ð¤Î°ÌÃÖ¤ò Ãµ¤¤Î¤Ë NetWare Nearest Server Query ¤ò»È¤¦Ù¤Ç¤¹¡£ ËÜ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤¬ false ¤Ç¤¢¤Ã¤¿¾ì¹ç¡¢¤â¤·¤¯¤Ï»ØÄê¤µ¤ì¤Ê¤«¤Ã¤¿¾ì¹ç¤Î Novell ¥¯¥é¥¤¥¢¥ó¥ÈÆ°ì¤Ïµ¬Äê¤µ¤ì¤Æ¤¤¤Þ¤»¤ó¡£

option nwip.preferred-dss *ip-address* [**,** *ip-address...*]**;**

ËÜ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤Ë¡¢5 ¤Ä¤Þ¤Ç¤Î IP ¥¢¥É¥ì¥¹¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¤½¤ì¤¾¤ì¤Î¥¢¥É¥ì¥¹¤Ï¡¢NetWare ¥É¥á¥¤¥ó SAP/RIP ¥µ¡¼¥Ð (DSS) ¤Î IP ¥¢¥É¥ì¥¹¤Ç¤¹¡£

option nwip.nearest-nwip-server *ip-address*
[**,** *ip-address...*]**;**

ËÜ¥µ¥Ö¥ª¥×¥·¥ç¥ó¤Ë¡¢5 ¤Ä¤Þ¤Ç¤Î IP ¥¢¥É¥ì¥¹¤Î¥ê¥¹¥È¤ò»ØÄê¤·¤Þ¤¹¡£ ¤½¤ì¤¾¤ì¤Î¥¢¥É¥ì¥¹¤Ï¡¢¶áÀÜ¤Î NetWare IP ¥µ¡¼¥Ð (Nearest NetWare IP Server) ¤Î IP ¥¢¥É¥ì¥¹¤Ç¤¹¡£

option nwip.autoretries *uint8***;**

µ¯Æ°»þ¤Ë¡¢NetWare/IP ¥¯¥é¥¤¥¢¥ó¥È¤¬¡¢Í¿¤¨¤é¤ì¤¿ DSS ¥µ¡¼¥Ð¤È ²¿²óÄÌ¿®¤ò»î¤ß¤ë¤Ù¤-¤«¤ò»ØÄê¤·¤Þ¤¹¡£

option nwip.autoretry-secs *uint8***;**

µ¯Æ°»þ¤Ë¡¢NetWare/IP ¥¯¥é¥¤¥¢¥ó¥È¤¬¡¢Í¿¤¨¤é¤ì¤¿ DSS ¥µ¡¼¥Ð¤È ÄÌ¿®¤ò³ÎÎ©¤¹¤ë»þ¤Ë¡¢¥ê¥È¥é¥¤¤Î´Ö²¿ÉÃÂÔ¤Ä¤Ù¤¤«¤ò»ØÄê¤·¤Þ¤¹¡£

option nwip.nwip-1-1 *uint8***;**

true ¤Ç¤¢¤Ã¤¿¾ì¹ç¡¢NetWare/IP ¥¯¥é¥¤¥¢¥ó¥È¤Ï NetWare/IP ¥Ð¡¼¥¸¥ç¥ó 1.1 ¤ò¥µ¥Ý¡¼¥È¤·¤Æ¤¤¤ë¤Ù¤¤Ç¤¹¡£ ¤³¤ì¡¢¥¯¥é¥¤¥¢¥ó¥È¤¬ NetWare/IP ¥Ð¡¼¥¸¥ç¥ó 1.1 ¤Î¥µ¡¼¥Ð¤È ÄÌ¿®¤¹¤ë»þ¤Î¤ßÉ¬Í×¤Ç¤¹¡£

option nwip.primary-dss *ip-address***;**

NetWare/IP ¥É¥á¥¤¥ó¤Î¥×¥é¥¤¥Þ¥ê¥É¥á¥¤¥ó SAP/RIP ¥µ¡¼¥Ó¥¹¥µ¡¼¥Ð (DSS) ¤Î IP ¥¢¥É¥ì¥¹¤ò»ØÄê¤·¤Þ¤¹¡£ ¥»¥«¥ó¥À¥ê DSS ¥µ¡¼¥Ð¤ÎÀßÄê»þ¤Ë¡¢NetWare/IP ´ÉÍý¥æ¡¼¥Æ¥£¥ê¥Æ¥£¤Ï¡¢ ¤³¤ÎÃÍ¤ò¥×¥é¥¤¥Þ¥ê DSS ¥µ¡¼¥Ð¤È¤·¤Æ»ÈÍÑ¤·¤Þ¤¹¡£

¿·µ¬¥ª¥×¥·¥ç¥ó¤ÎÄêµÁ

Internet Systems Consortium DHCP ¥¯¥é¥¤¥¢¥ó¥È¤È¥µ¡¼¥Ð¤Ï¡¢ ¿·µ¬¥ª¥×¥·¥ç¥ó¤òÄêµÁ¤¹¤ëµ¡¹½¤âÄó¶¡¤·¤Æ¤¤¤Þ¤¹¡£ ¤½¤ì¤¾¤ì¤Î DHCP ¥ª¥×¥·¥ç¥ó¤Ï¡¢Ì¾Á°¤È¥³¡¼¥É¡¢¹½Â¤¤ò»ý¤Ã¤Æ¤¤¤Þ¤¹¡£ Ì¾Á°¤Ï¡¢»ÈÍÑ¼Ô¬¥ª¥×¥·¥ç¥ó¤ò»²¾È¤¹¤ë¤Î¤Ë»ÈÍÑ¤µ¤ì¤Þ¤¹¡£ ¥³¡¼¥É¤Ï¡¢DHCP ¥µ¡¼¥Ð¤È¯¥é¥¤¥¢¥ó¥È¬¥ª¥×¥·¥ç¥ó¤ò»²¾È¤¹¤ë¤Î¤Ë »ÈÍÑ¤¹¤ëÖ¹æ¤Ç¤¹¡£ ¹½Â¤¤Ï¡¢ª¥×¥·¥ç¥ó¤ÎÆâÍÆ¤¬É¤Î¤è¤¦¤Ê¤â¤Î¤«¤òµ½Ò¤·¤Æ¤¤¤Þ¤¹¡£

¿·µ¬¥ª¥×¥·¥ç¥ó¤òÄêµÁ¤¹¤ë¤Ë¤Ï¡¢Â¾¤Î¥ª¥×¥·¥ç¥ó¤Ç¤Ï»È¤ï¤ì¤Æ¤¤¤Ê¤¤Ì¾Á°¤ò Á ª¤ÖÉ¬Í×¤¬¤¢¤ê¤Þ¤¹¡£ Îã¤¨¤Ð¡¢"host-name" ¤È¸À¤¦Ì¾Á°¤Ï»ÈÍÑ¤Ç¤¤Þ¤»¤ó¡£ ¤È¤¤¤¦¤Î¤â¡¢¤³¤Î¥Þ¥Ë¥å¥¢¥ë¥Ú¡¼¥¸Ë½Ð¤Æ¤¤¿¤è¦¤Ë¡¢ DHCP ¥×¥í¥È¥³¥ë¬´û¤Ë host-name ¥ª¥×¥·¥ç¥ó¤òÄêµÁ¤·¤Æ¤¤¤ë¤«¤é¤Ç¤¹¡£ ¤³¤Î¥Þ¥Ë¥å¥¢¥ë¥Ú¡¼¥¸Ë½Ð¤Æ¤¤ ¤Æ¤¤¤Ê¤¤¥ª¥×¥·¥ç¥óÌ¾¤Ê¤é¤Ð »È¤Ã¤Æ¤â¹½¤¤Þ¤»¤ó¤¬¡¢¾¾Íè½Ð¤Æ¤¯¤ë¥ª¥×¥·¥ç¥ó¤È½Å¤Ê¤é¤Ê¤¤¤è¦¤Ë¡¢ ¥ª¥×¥·¥ç¥óÌ¾¤ÎºÇ½é¤ËÆÈ¼¤ÎÊ¸»úÎó¤ò¤Ä¤±¤ë¤³¤È¤Ï¡¢Â¿Ê¬¤¤¤¤¹Í¤¨¤Ç¤·¤ç¤¦¡£ Îã¤¨¤Ð¡¢¸ø¼°¤Î DHCP ¥ª¥×¥·¥ç¥ó¤Ë¤Ï "local" ¤Ç»Ï¤Þ¤ë¤â¤Î¤¬¤Ê¤¤¤Î¤Ç¡¢ "local-host-name" ¤È¸À¤¦Ì¾Á°¤Ï¡¢¤¤¤¯¤é¤«°Â¿´¤·¤ÆÄêµÁ¤Ç¤¤ë¤Ç¤·¤ç¤¦¡£

Ì¾Á°¤òÁªÂò¤·¤¿¤é¡¢¼Ï¥³¡¼¥É¤òÁª¤Ð¤Í¤Ð¤Ê¤êÞ¤»¤ó¡£ DHCP ¥ª¥×¥·¥ç¥ó¤Î 128 ¤«¤é 256 ¤Þ¤Ç¤Î¥³¡¼¥É¤Ï¡¢ ¥µ¥¤È¥í¼¥«¥ë¥ª¥×¥·¥ç¥óÍÑ¤È¤·¤ÆÍ½Ìó¤µ¤ì¤Æ¤¤¤ë¤Î¤Ç¡¢ ¤³¤ÎÃæ¤Î¥³¡¼¥É¤Ê¤é¤É¤ì¤Ç¤âÁª¤Ö¤³¤È¬¤Ç¤¤Þ¤¹¡£ ¼ÂºÝ¤Ë¤Ï¡¢¥×¥íÈ¥³¥ë¤ò¾¯¡¤¢¤¤¤Þ¤¤¤Ë²ò¼á¤·¤Æ¤¤¤ë¥Ù¥ó¥À¬¬¤¢¤ê¡¢ 128 ¤è¤êÂç¤¤¤ÃÍ¤Î¥ª¥×¥·¥ç¥ó¥³¡¼¥É¤ò»ÈÍÑ¤·¤Æ¤¤¤Þ¤¹¡£ ¤³¤ÎÌäÂê¤òËÜÅö¤Ë²óÈò¤¹¤ëÊý¤¡¤Ï¤¢¤ê¤Þ¤»¤ó¤¬¡¢ ¼ÂºÝ¤Ë¤Ï¤½¤¦Â¿¤¤ÊÌäÂê¤ò°ú¤µ¯¤³¤¹¤â¤Î¤Ç¤Ï¤Ê¤¤¤Ç¤·¤ç¤¦¡£

¥ª¥×¥·¥ç¥ó¤Î¹½Â¤¤È¡¢Ã±Ë¥ª¥×¥·¥ç¥óÎ¥Ç¡¼¥¿¬¤É½¸¤µ¤ì¤Æ¤¤¤ë·Á¼°¤Ç¤¹¡£ ¸½ºß ISC DHCP ¥µ¡¼¥Ð¤Ï¡¢À°¿ô¡¢¥Ö¡¼¥ëÃÍ¡¢Ê¸»úÎó¤½¤·¤Æ IP ¥¢¥É¥ì¥¹¤È¤¤¤Ã¤¿¡¢ ¤¤¤¯¤Ä¤«¤ÎÃ±½ã¤Ê¥Ç¡¼¥¿¿·¿ò¥µ¥Ý¡¼¥È¤·¤Æ¤ª¤ê¡¢ ¤Þ¤¿Ã±°ì¥Ç¡¼¥¿¿·¿¤ÎÇÛÎó¤ä¸ÇÄê½ç¤Î¥Ç¡¼¥¿¿·¿ÎÇÛÎó¤òÄêµÁ¤¹¤ë¤³¤È¤â¤Ç¤¤Þ¤¹¡£

¿·µ¬¥ª¥×¥·¥ç¥ó¤Ï¡¢°Ê²¼¤Î¤è¦¤ËÀë¸À¤µ¤ì¤Þ¤¹:

option *new-name* **code** *new-code* **=** *definition* **;**

new-name ¤È *new-code* ¤ÎÃÍ¤Ï¡¢¿·µ¬¥ª¥×¥·¥ç¥óÍÑ¤Ë¤¢¤Ê¤¿¤¬Á ª¤ó¤À¤â¤Î¤Ç¤¹¡£ *definition* ¤Ï¡¢¥ª¥×¥·¥ç¥ó¤Î¹½Â¤¤ÎÄêµÁ¤Ç¤¹¡£

°Ê²¼¤ÎÃ±½ã¤Ê¥ª¥×¥·¥ç¥ó¤Î·¿ÄêµÁ¤¬¥µ¥Ý¡¼¥È¤µ¤ì¤Æ¤¤¤Þ¤¹:

¥Ö¡¼¥ëÃÍ

option *new-name* **code** *new-code* **= boolean ;**

¥Ö¡¼¥ë·¿¤Î¥ª¥×¥·¥ç¥ó¤Ï¡¢on ¤Þ¤¿¤Ï off (¤â¤·¤¯¤Ï true ¤« false) ¤ÎÃÍ¤ò »ý¤Ä¥Õ¥é¥°¤Ç¤¹¡£ ¥Ö¡¼¥ë·¿¤Î»ÈÍÑÎã¤Ï¡¢°Ê²¼¤Î¤è¦¤ËÊ¸»úÎó¤Ë¤Ê¤ê¤Þ¤¹:

```
option use-zephyr code 180 = boolean;
option use-zephyr on;
```

À°¿ô

option *new-name* **code** *new-code* **=** *sign* **integer** *width* **;**

sign ¥È¡¼¥¯¤Ï¡¢¶õÇò¡¢*unsigned* ¡¢*signed* ¤Î¤¤º¤ì¤«¤Ç¤¹¡£ width ¤Ï 8, 16, 32 ¤Î¤¤º¤ì¤«¤Ç¡¢À°¿ô¤Î bit ¿ô¤ò¼¨¤·¤Þ¤¹¡£ Îã¤¨¤Ð¡¢°Ê²¼¤Î 2 ¹Ô¤Ï¡¢sql-connection-max ¥ª¥×¥·¥ç¥óÎÄêµÁ¤È »ÈÍÑ¡¢¤ò¼¨¤·¤Þ¤¹:

```
option sql-connection-max code 192 = unsigned integer 16;
```

option sql-connection-max 1536;

IP ¥¢¥É¥ì¥¹

option *new-name* **code** *new-code* **= ip-address ;**

IP ¥¢¥É¥ì¥¹·¿¤Î½Â¤¤ò»ý¤Ä¥ª¥×¥·¥ç¥ó¤Ï¡¢¥É¥á¥¤¥óÌ¾¤â¤·¤¯¤Ï ¥É¥Ã¥È¶èÀÚ¤ê¤Î 4 À°¿ô¤ÇÉ½¸½¤µ¤ì¤Þ¤¹¡£ °Ê²¼¤Ï¡¢IP ¥¢¥É¥ì¥¹·¿¤Î»ÈÍÑÎã¤Ç¤¹:

option sql-server-address code 193 = ip-address;
option sql-server-address sql.example.com;

¥Æ¥¥¹¥È

option *new-name* **code** *new-code* **= text ;**

¥Æ¥¥¹¥È·¿¤Î¥ª¥×¥·¥ç¥ó¤Ï¡¢ASCII ¥Æ¥¥¹¥È¸¸»úÎó¤ò¥¨¥ó¥³¡¼¥É¤·¤Þ¤¹¡£ Îã¤¨¤Ð:

option sql-default-connection-name code 194 = text;
option sql-default-connection-name "PRODZA";

¥Ç¡¼¥¿¸Ê¸»úÎó

option *new-name* **code** *new-code* **= string ;**

¥Ç¡¼¥¿¸Ê¸»úÎó·¿¤Î¥ª¥×¥·¥ç¥ó¤Ï¡¢ËÜ¼ÁÅª¤Ë¤ÏÃ±¤Ê¤ë¥Ð¥¤¥È¤Î½¸¹çÂÎ¤Ç¤¹¡£ ¥Æ¥-¥¹¥È·¿¤Î¤è¤¦¤Ë¥¯¥ª¡¼¥È¤µ¤ì¤¿¥Æ¥¥¹¥È¤Ç»ØÄê¤µ¤ì¤«¡¢ ¤â¤·¤¯¤Ï¥³¥í¥¶èÀÚ¤ê¤Î 16 ¿Ê¿ô¤Î¥ê¥¹¥È¤Ç»ØÄê¤µ¤ì¤Þ¤¹¡£ ¤³¤Î¥³¥í¥Ç¶èÀÚ¤é¤ì¤¿Ãæ¿È¤Ï¡¢0 ¤«¤é FF ¤Î´Ö¤ÎÃÍ¤Ç¤Ê¤±¤ì¤Ð¤Ê¤ê¤Þ¤»¤ó¡£ Îã¤¨¤Ð:

option sql-identification-token code 195 = string;
option sql-identification-token 17:23:19:a6:42:ea:99:7c:22;

¥«¥×¥»¥ë²½

option *new-name* **code** *new-code* **= encapsulate** *identifier* **;**

¥«¥×¥»¥ë²½·¿¤Î¥ª¥×¥·¥ç¥ó¤Ï¡¢*identifier* ¤Ç»ØÄê¤µ¤ì¤¿ ¥ª¥×¥·¥ç¥ó¶õ´Ö¤ÎÃæ¿È¤ò¥«¥×¥»¥ë²½¤·¤Þ¤¹¡£ ¸½ºß DHCP ¥×¥í¥È¥³¥ë¤ËÂ¸ºß¤¹¤ë¥«¥×¥»¥ë²½¥ª¥×¥·¥ç¥ó¤ÎÎã¤Ï¡¢ vendor-encapsulated-options ¥ª¥×¥·¥ç¥ó¡¢netware-suboptions ¥ª¥×¥·¥ç¥ó¡¢ relay-agent-information ¥ª¥×¥·¥ç¥ó¤Ê¤É¤Ç¤¹¡£

option space local;
option local.demo code 1 = text;
option local-encapsulation code 197 = encapsulate local;
option local.demo "demo";

ÇÛÎó

¥ª¥×¥·¥ç¥ó¤Ï¡¢¥Æ¥¥¹¥È·¿¤È¥Ç¡¼¥¿¸Ê¸»úÎó·¿°Ê³°¤Î¾å½Ò¤Î¤¤¤«¤Ê¤ë¥Ç¡¼¥¿·¿¤Î ÇÛÎó¤â´Þ¤à¤³¤È¤¬Ç¤¤Þ¤¹¡£ ¥Æ¥-¥¹¥È·¿¤È¥Ç¡¼¥¿¸Ê¸»úÎó·¿¤Ï¡¢½ºßÇÛÎó¤Ç¤Ï¥µ¥Ý¡¼¥È¤µ¤ì¤Æ¤¤¤Þ¤»¤ó¡£ ÇÛÎóÄêµÁ¤ÎÎã¤Ï°Ê²¼¤ÎÀÌ¤ê¤Ç¤¹:

option kerberos-servers code 200 = array of ip-address;
option kerberos-servers 10.20.10.1, 10.20.11.1;

¥ì¥³¡¼¥É

¥ª¥×¥·¥ç¥ó¤Ï¡¢¥Ç¡¼¥¿·¿¤ÎÎó¤Ç¹½À®¤µ¤ì¤ë¥Ç¡¼¥¿¸½Â¤¤ò´Þ¤à¤³¤È¤â¤Ç¤¤Þ¤¹¡£

¤³¤ì¤Ï¤·¤Ð¤·¤Ð¥ì¥³¡¼¥É·¿¤È¸Æ¤Ð¤ì¤Þ¤¹¡£ Îã¤¨¤Ð:

```
option contrived-001 code 201 = { boolean, integer 32, text };
option contrived-001 on 1772 "contrivance";
```

¤Þ¤¿¥ì¥³¡¼¥ÉºÎÇÛÎó¤Î¥ª¥×¥·¥ç¥ó¤ò»ý¤Ä¤³¤È¤â¤Ç¤¤Þ¤¹¡£ Îã¤¨¤Ð:

```
option new-static-routes code 201 = array of {
        ip-address, ip-address, ip-address, integer 8 };
option static-routes
        10.0.0.0 255.255.255.0 net-0-rtr.example.com 1,
        10.0.1.0 255.255.255.0 net-1-rtr.example.com 1,
        10.2.0.0 255.255.224.0 net-2-0-rtr.example.com 3;
```

¥Ù¥ó¥À¥«¥×¥»¥ë²½¥ª¥×¥·¥ç¥ó

DHCP ¥×¥í¥È¥³¥ë¤Ë¤Ï¡¢ **vendor-encapsulated-options** ¥ª¥×¥·¥ç¥ó¤¬ ÄêµÁ¤µ¤ì¤Æ¤¤¤Þ¤¹¡£ ¥Ù¥ó¥À¤Ï¡¢¤³¤Î¥ª¥×¥·¥ç¥ó¤Ë¤è¤Ã¤Æ¡¢¥Ù¥ó¥À¸ÇÍ¤Î¥ª¥×¥·¥ç¥ó¤ò É¸½à DHCP ¥ª¥×¥·¥ç¥ó¤Ë´Þ¤á¤ÆÁ÷½Ð¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ **vendor-encapsulated-options** ¥ª¥×¥·¥ç¥ó¤Î½ñ¼°¤Ï¡¢½ñ¼°¤¬µ¬Äê¤µ¤ì¤Æ¤¤¤Ê¤¤°¤ìÏ¢¤Î¥Ð¥¤¥È¡¢ ¤â¤·¯¤Ï°ìÏ¢¤Î¥ª¥×¥·¥ç¥óÎó¤Ç¤¹¡£ ¥ª¥×¥·¥ç¥óÎóÃæ¤Î¤½¤ì¾¤ì¤Î¥ª¥×¥·¥ç¥ó¤Ï¡¢1 ¥Ð¥¤¥È¤Î¥Ù¥ó¥À¸ÇÍ¤Î ¥ª¥×¥·¥ç¥ó¥¡¼¥É¤Î¸å¤Ë 1 ¥Ð¥¤¥È¤Î¥Ç¡¼¥¿¡¢ ¤½¤·¤Æ¤Î¥Ç¡¼¥¿¤¬¤Ä¹«¤Ê¤¤¥×¥·¥ç¥ó¥³¡¼¥É¤Ï´Þ¤Þ¤ì¤Þ¤»¤ó)¡£

ËÜ¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤Ï¡¢2 ¤Ä¤ÎÊýË¡¤Î¤¤º¤ì¤«¤ÇÀßÄê¤µ¤ì¤Þ¤¹¡£ 1 ÈÖÌÜ¤ÎÊýË¡¤Ï¡¢Ã±¤Ë¥Ç¡¼¥¿¤òÄ¾ÀÜ»ØÄê¤¹¤ë¤â¤Î¤Ç¤¹¡£ ¥Ç¡¼¥¿¤Î»ØÄê¤Ë¤Ï¡¢¥Æ¥¥¹¥ÈÊ¸»úÎó¤«¥³¥í¥ó¤Ç¶èÀÚ¤é¤ì¤¿ 16 ¿Ê¿ôÃÍ¤òÍÑ¤¤¤Þ¤¹¡£ Îã¤¨¤Ð:

```
option vendor-encapsulated-options
  2:4:AC:11:41:1:
  3:12:73:75:6e:64:68:63:70:2d:73:65:72:76:65:72:31:37:2d:31:
  4:12:2f:65:78:70:6f:72:74:2f:72:6f:6f:74:2f:69:38:36:70:63;
```

ËÜ¥ª¥×¥·¥ç¥ó¤òÀßÄê¤¹¤ë 2 ÈÖÌÜ¤ÎÊýË¡¤Ï¡¢DHCP ¥µ¡¼¥Ð¤Ë ¥Ù¥ó¥À¸ÇÍ¥ª¥×¥·¥ç¥ó¥Ð¥Ã¥Õ¥¡¤òºîÀ®¤µ»¤ë¤È¤¤¦¤â¤Î¤Ç¤¹¡£ ¤³¤ì¤ò¤¹¤ë¤Ë¤Ï¡¢°Ê²¼¤Î 4 ¤Ä¤Î¤³¤È¤ò¤¹¤ëÉ¬Í×¤¬¤¢¤ê¤Þ¤¹:

¥ª¥×¥·¥ç¥ó¶õ´Ö¤òÄêµÁ¤·¡¢¤½¤Î¥ª¥×¥·¥ç¥ó¶õ´ÖÆâ¤Ë¥ª¥×¥·¥ç¥ó¤òÄêµÁ¤·¡¢
¤½¤ì¤é¤ØÃÍ¤ò³ä¤ê¿¶¤ê¡¢¤ºÇ¸å¤Ë¤½¤Î¥ª¥×¥·¥ç¥ó¶õ´Ö¤¬ **vendor-encapsulated-options** ¥ª¥×¥·¥ç¥ó¤ÎÀ¸À®¤Ë»ÈÍÑ¤µ¤ì¤³¤È¤ò»ØÄê¤·¤Þ¤¹¡£

¥Ù¥ó¥À¥ª¥×¥·¥ç¥ó¤¬¾ÊÇ¼¤µ¤ì¤ë¥ª¥×¥·¥ç¥ó¶õ´Ö¤ò¿·µ¬¤ËÄêµÁ¤¤¤ë¤Ë¡¢ option space Ê¸¤ò»ÈÍÑ¤·¤Þ¤¹:

option space *name* **;**

¤³¤ÎÊ¸½ñ¤Ë¤³¤ì¤Æ¤¤¤ë¤è¤¦¤Ë¡¢ ¤³¤Î name ¤Ï¡¢¥ª¥×¥·¥ç¥óÄêµÁ¤Ç»ÈÍÑ¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ Îã¤¨¤Ð:

```
option space SUNW;
option SUNW.server-address code 2 = ip-address;
option SUNW.server-name code 3 = text;
option SUNW.root-path code 4 = text;
```

°ìÅÙ¡¢¥ª¥×¥·¥ç¥ó¶õ´Ö¤È¥ª¥×¥·¥ç¥ó¤Î½ñ¼°¤òÄêµÁ¤·¤¿¤é¡¢
¤½¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤òÄêµÁ¤¹¤ë¥¹¥³¡¼¥×¤òÀßÄê¤Ç¤¡¢
¤½¤ì¤é¤Î¥ª¥×¥·¥ç¥ó¤ò¤¤¤Ä»Ë¤¦¤«¤ò»ØÄê¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£ Îã¤¨¤Ð¡¢2 ¤Ä¤Î°Û¤Ê¤ë¥¯¥é¥¹¤Î¥¯¥é¥¢¥¤¥ó¥È¤ò°·¤¤¿¤¤È¤·¤Þ¤·¤ç¤¦¡£
Á°½Ò¤ÎÎã¤Ç¼¨¤·¤¿¥ª¥×¥·¥ç¥ó¶õ´ÖÄêµÁ¤ò»È¤Ã¤Æ¡¢°Ê²¼¤Î¤è¤¦¡£

¥¯¥é¥¤¥¢¥ó¥È¤«¤éÁ÷¤é¤ì¤Æ¤¤¿ vendor-class-identifier ¥ª¥×¥·¥ç¥ó¤Ë´ð¤Å¤¤¤Æ¡¢°Û¤Ê¤ë¥ª¥×¥·¥ç¥ó¤ÎÃÍ¤ò°Û¤Ê¤ë¥¯¥é¥¤¥¢¥ó¥È¤ËÁ÷½Ð¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£

```
class "vendor-classes" {
  match option vendor-class-identifier;
}

option SUNW.server-address 172.17.65.1;
option SUNW.server-name "sundhcp-server17-1";

subclass "vendor-classes" "SUNW.Ultra-5_10" {
  vendor-option-space SUNW;
  option SUNW.root-path "/export/root/sparc";
}

subclass "vendor-classes" "SUNW.i86pc" {
  vendor-option-space SUNW;
  option SUNW.root-path "/export/root/i86pc";
}
```

Àè¤ÎÎã¤Ç¸«¤¿¤è¤¦¤Ë¡¢ÄÌ¾ï¤Î¥¹¥¿¼×¥ë¡¼¥ë¤òÅ¬ÍÑ¤¹¤ë¤³¤È¤Ç¡¢¥°¥í¡¼¥Ð¥ë¤ÊÃÍ¤ò¥°¥í¡¼¥Ð¥ë¥¹¥¿¡¼¥×Ãæ¤ËÄêµÁ¤Ç¤¡¢ ÆÃÄê¤Î¥¯¥é¥¤¥Ë¸ÇÍ¤ÎÃÍ¤À¤±¤ò¥í¡¼¥«¥ë¥¹¥¿¡¼¥×¤ËÄêµÁ¤Ç¤¤Þ¤¹¡£ **vendor-option-space** Àë¸À¤ò»È¤¦¤³¤È¤Ç¡¢ **vendor-encapsulated-options** ¥ª¥×¥·¥ç¥ó¤ò¹½À®¤¹¤ë¤Î¤Ë¡¢SUNW ¥ª¥×¥·¥ç¥ó¶õ´ÖÆâ¤Î¥ª¥×¥·¥ç¥ó¤ò»È¤¦¤è¤¦DHCP ¥µ¡¼¥Ð¤Ë»Ø¼¨¤¹¤ë¤³¤È¤¬¤Ç¤¤Þ¤¹¡£

´ØÏ¢¹àÌÜ

dhclient.conf(5), dhcp-eval(5), dhclient(8), RFC2132, RFC2131

ºî¼Ô

Internet Systems Consortium DHCP Distribution ¤Ï¡¢Vixie Labs ¤È¤Î·ÀÌó¤Î¤â¤È¤Ç¡¢Ted Lemon ¤¬µ-½Ò¤·¤Þ¤·¤¿¡£ ËÜ¥×¥í¥¸§¯¥È¤Î»ñ¶â¤Ï¡¢Internet Systems Consortium ¤¬Äó¶¡¤·¤Þ¤·¤¿¡£ Internet Systems Consortium ¤Ë´Ø¤¹¤ë¾ðÊó¤Ï¡¢ **http://www.isc.org** ¤Ë¤¢¤ê¤Þ¤¹¡£

NAME

dhcpd.conf - dhcpd configuration file

DESCRIPTION

The dhcpd.conf file contains configuration information for *dhcpd*, the Internet Systems Consortium DHCP Server.

The dhcpd.conf file is a free-form ASCII text file. It is parsed by the recursive-descent parser built into dhcpd. The file may contain extra tabs and newlines for formatting purposes. Keywords in the file are case-insensitive. Comments may be placed anywhere within the file (except within quotes). Comments begin with the # character and end at the end of the line.

The file essentially consists of a list of statements. Statements fall into two broad categories - parameters and declarations.

Parameter statements either say how to do something (e.g., how long a lease to offer), whether to do something (e.g., should dhcpd provide addresses to unknown clients), or what parameters to provide to the client (e.g., use gateway 220.177.244.7).

Declarations are used to describe the topology of the network, to describe clients on the network, to provide addresses that can be assigned to clients, or to apply a group of parameters to a group of declarations. In any group of parameters and declarations, all parameters must be specified before any declarations which depend on those parameters may be specified.

Declarations about network topology include the *shared-network* and the *subnet* declarations. If clients on a subnet are to be assigned addresses dynamically, a *range* declaration must appear within the *subnet* declaration. For clients with statically assigned addresses, or for installations where only known clients will be served, each such client must have a *host* declaration. If parameters are to be applied to a group of declarations which are not related strictly on a per-subnet basis, the *group* declaration can be used.

For every subnet which will be served, and for every subnet to which the dhcp server is connected, there must be one *subnet* declaration, which tells dhcpd how to recognize that an address is on that subnet. A *subnet* declaration is required for each subnet even if no addresses will be dynamically allocated on that subnet.

Some installations have physical networks on which more than one IP subnet operates. For example, if there is a site-wide requirement that 8-bit subnet masks be used, but a department with a single physical ethernet network expands to the point where it has more than 254 nodes, it may be necessary to run two 8-bit subnets on the same ethernet until such time as a new physical network can be added. In this case, the *subnet* declarations for these two networks must be enclosed in a *shared-network* declaration.

Some sites may have departments which have clients on more than one subnet, but it may be desirable to offer those clients a uniform set of parameters which are different than what would be offered to clients from other departments on the same subnet. For clients which will be declared explicitly with *host* declarations, these declarations can be enclosed in a *group* declaration along with the parameters which are common to that department. For clients whose addresses will be dynamically assigned, class declarations and conditional declarations may be used to group parameter assignments based on information the client sends.

When a client is to be booted, its boot parameters are determined by consulting that client's *host* declaration (if any), and then consulting any *class* declarations matching the client, followed by the *pool*, *subnet* and *shared-network* declarations for the IP address assigned to the client. Each of these declarations itself appears within a lexical scope, and all declarations at less specific lexical scopes are also consulted for client option declarations. Scopes are never considered twice, and if parameters are declared in more than one scope, the parameter declared in the most specific scope is the one that is used.

When dhcpd tries to find a *host* declaration for a client, it first looks for a *host* declaration which has a *fixed-address* declaration that lists an IP address that is valid for the subnet or shared network on which the client is booting. If it doesn't find any such entry, it tries to find an entry which has no *fixed-address* declaration.

EXAMPLES

A typical dhcpd.conf file will look something like this:

```
global parameters...

subnet 204.254.239.0 netmask 255.255.255.224 {
  subnet-specific parameters...
  range 204.254.239.10 204.254.239.30;
}

subnet 204.254.239.32 netmask 255.255.255.224 {
  subnet-specific parameters...
  range 204.254.239.42 204.254.239.62;
}

subnet 204.254.239.64 netmask 255.255.255.224 {
  subnet-specific parameters...
  range 204.254.239.74 204.254.239.94;
}

group {
  group-specific parameters...
  host zappo.test.isc.org {
    host-specific parameters...
  }
  host beppo.test.isc.org {
    host-specific parameters...
  }
  host harpo.test.isc.org {
    host-specific parameters...
  }
}
```

Figure 1

Notice that at the beginning of the file, there's a place for global parameters. These might be things like the organization's domain name, the addresses of the name servers (if they are common to the entire organization), and so on. So, for example:

```
option domain-name "isc.org";
option domain-name-servers ns1.isc.org, ns2.isc.org;
```

Figure 2

As you can see in Figure 2, you can specify host addresses in parameters using their domain names rather than their numeric IP addresses. If a given hostname resolves to more than one IP address (for example, if that host has two ethernet interfaces), then where possible, both addresses are supplied to the client.

The most obvious reason for having subnet-specific parameters as shown in Figure 1 is that each subnet, of necessity, has its own router. So for the first subnet, for example, there should be something like:

```
option routers 204.254.239.1;
```

Note that the address here is specified numerically. This is not required - if you have a different domain name for each interface on your router, it's perfectly legitimate to use the domain name for that interface instead of the numeric address. However, in many cases there may be only one domain name for all of a

router's IP addresses, and it would not be appropriate to use that name here.

In Figure 1 there is also a *group* statement, which provides common parameters for a set of three hosts - zappo, beppo and harpo. As you can see, these hosts are all in the test.isc.org domain, so it might make sense for a group-specific parameter to override the domain name supplied to these hosts:

```
option domain-name "test.isc.org";
```

Also, given the domain they're in, these are probably test machines. If we wanted to test the DHCP leasing mechanism, we might set the lease timeout somewhat shorter than the default:

```
max-lease-time 120;
default-lease-time 120;
```

You may have noticed that while some parameters start with the *option* keyword, some do not. Parameters starting with the *option* keyword correspond to actual DHCP options, while parameters that do not start with the option keyword either control the behavior of the DHCP server (e.g., how long a lease dhcpd will give out), or specify client parameters that are not optional in the DHCP protocol (for example, server-name and filename).

In Figure 1, each host had *host-specific parameters*. These could include such things as the *hostname* option, the name of a file to upload (the *filename* parameter) and the address of the server from which to upload the file (the *next-server* parameter). In general, any parameter can appear anywhere that parameters are allowed, and will be applied according to the scope in which the parameter appears.

Imagine that you have a site with a lot of NCD X-Terminals. These terminals come in a variety of models, and you want to specify the boot files for each model. One way to do this would be to have host declarations for each server and group them by model:

```
group {
  filename "Xncd19r";
  next-server ncd-booter;

  host ncd1 { hardware ethernet 0:c0:c3:49:2b:57; }
  host ncd4 { hardware ethernet 0:c0:c3:80:fc:32; }
  host ncd8 { hardware ethernet 0:c0:c3:22:46:81; }
}

group {
  filename "Xncd19c";
  next-server ncd-booter;

  host ncd2 { hardware ethernet 0:c0:c3:88:2d:81; }
  host ncd3 { hardware ethernet 0:c0:c3:00:14:11; }
}

group {
  filename "XncdHMX";
  next-server ncd-booter;

  host ncd1 { hardware ethernet 0:c0:c3:11:90:23; }
  host ncd4 { hardware ethernet 0:c0:c3:91:a7:8; }
  host ncd8 { hardware ethernet 0:c0:c3:cc:a:8f; }
}
```

ADDRESS POOLS

The **pool** declaration can be used to specify a pool of addresses that will be treated differently than another pool of addresses, even on the same network segment or subnet. For example, you may want to provide a

large set of addresses that can be assigned to DHCP clients that are registered to your DHCP server, while providing a smaller set of addresses, possibly with short lease times, that are available for unknown clients. If you have a firewall, you may be able to arrange for addresses from one pool to be allowed access to the Internet, while addresses in another pool are not, thus encouraging users to register their DHCP clients. To do this, you would set up a pair of pool declarations:

```
subnet 10.0.0.0 netmask 255.255.255.0 {
  option routers 10.0.0.254;

  # Unknown clients get this pool.
  pool {
    option domain-name-servers bogus.example.com;
    max-lease-time 300;
    range 10.0.0.200 10.0.0.253;
    allow unknown-clients;
  }

  # Known clients get this pool.
  pool {
    option domain-name-servers ns1.example.com, ns2.example.com;
    max-lease-time 28800;
    range 10.0.0.5 10.0.0.199;
    deny unknown-clients;
  }
}
```

It is also possible to set up entirely different subnets for known and unknown clients - address pools exist at the level of shared networks, so address ranges within pool declarations can be on different subnets.

As you can see in the preceding example, pools can have permit lists that control which clients are allowed access to the pool and which aren't. Each entry in a pool's permit list is introduced with the *allow* or *deny* keyword. If a pool has a permit list, then only those clients that match specific entries on the permit list will be eligible to be assigned addresses from the pool. If a pool has a deny list, then only those clients that do not match any entries on the deny list will be eligible. If both permit and deny lists exist for a pool, then only clients that match the permit list and do not match the deny list will be allowed access.

DYNAMIC ADDRESS ALLOCATION

Address allocation is actually only done when a client is in the INIT state and has sent a DHCPDISCOVER message. If the client thinks it has a valid lease and sends a DHCPREQUEST to initiate or renew that lease, the server has only three choices - it can ignore the DHCPREQUEST, send a DHCPNAK to tell the client it should stop using the address, or send a DHCPACK, telling the client to go ahead and use the address for a while.

If the server finds the address the client is requesting, and that address is available to the client, the server will send a DHCPACK. If the address is no longer available, or the client isn't permitted to have it, the server will send a DHCPNAK. If the server knows nothing about the address, it will remain silent, unless the address is incorrect for the network segment to which the client has been attached and the server is authoritative for that network segment, in which case the server will send a DHCPNAK even though it doesn't know about the address.

There may be a host declaration matching the client's identification. If that host declaration contains a fixed-address declaration that lists an IP address that is valid for the network segment to which the client is connected. In this case, the DHCP server will never do dynamic address allocation. In this case, the client is *required* to take the address specified in the host declaration. If the client sends a DHCPREQUEST for some other address, the server will respond with a DHCPNAK.

When the DHCP server allocates a new address for a client (remember, this only happens if the client has sent a DHCPDISCOVER), it first looks to see if the client already has a valid lease on an IP address, or if there is an old IP address the client had before that hasn't yet been reassigned. In that case, the server will

take that address and check it to see if the client is still permitted to use it. If the client is no longer permitted to use it, the lease is freed if the server thought it was still in use - the fact that the client has sent a DHCPDISCOVER proves to the server that the client is no longer using the lease.

If no existing lease is found, or if the client is forbidden to receive the existing lease, then the server will look in the list of address pools for the network segment to which the client is attached for a lease that is not in use and that the client is permitted to have. It looks through each pool declaration in sequence (all *range* declarations that appear outside of pool declarations are grouped into a single pool with no permit list). If the permit list for the pool allows the client to be allocated an address from that pool, the pool is examined to see if there is an address available. If so, then the client is tentatively assigned that address. Otherwise, the next pool is tested. If no addresses are found that can be assigned to the client, no response is sent to the client.

If an address is found that the client is permitted to have, and that has never been assigned to any client before, the address is immediately allocated to the client. If the address is available for allocation but has been previously assigned to a different client, the server will keep looking in hopes of finding an address that has never before been assigned to a client.

The DHCP server generates the list of available IP addresses from a hash table. This means that the addresses are not sorted in any particular order, and so it is not possible to predict the order in which the DHCP server will allocate IP addresses. Users of previous versions of the ISC DHCP server may have become accustomed to the DHCP server allocating IP addresses in ascending order, but this is no longer possible, and there is no way to configure this behavior with version 3 of the ISC DHCP server.

IP ADDRESS CONFLICT PREVENTION

The DHCP server checks IP addresses to see if they are in use before allocating them to clients. It does this by sending an ICMP Echo request message to the IP address being allocated. If no ICMP Echo reply is received within a second, the address is assumed to be free. This is only done for leases that have been specified in range statements, and only when the lease is thought by the DHCP server to be free - i.e., the DHCP server or its failover peer has not listed the lease as in use.

If a response is received to an ICMP Echo request, the DHCP server assumes that there is a configuration error - the IP address is in use by some host on the network that is not a DHCP client. It marks the address as abandoned, and will not assign it to clients.

If a DHCP client tries to get an IP address, but none are available, but there are abandoned IP addresses, then the DHCP server will attempt to reclaim an abandoned IP address. It marks one IP address as free, and then does the same ICMP Echo request check described previously. If there is no answer to the ICMP Echo request, the address is assigned to the client.

The DHCP server does not cycle through abandoned IP addresses if the first IP address it tries to reclaim is free. Rather, when the next DHCPDISCOVER comes in from the client, it will attempt a new allocation using the same method described here, and will typically try a new IP address.

DHCP FAILOVER

This version of the ISC DHCP server supports the DHCP failover protocol as documented in draft-ietf-dhc-failover-07.txt. This is not a final protocol document, and we have not done interoperability testing with other vendors' implementations of this protocol, so you must not assume that this implementation conforms to the standard. If you wish to use the failover protocol, make sure that both failover peers are running the same version of the ISC DHCP server.

The failover protocol allows two DHCP servers (and no more than two) to share a common address pool. Each server will have about half of the available IP addresses in the pool at any given time for allocation. If one server fails, the other server will continue to renew leases out of the pool, and will allocate new addresses out of the roughly half of available addresses that it had when communications with the other server were lost.

It is possible during a prolonged failure to tell the remaining server that the other server is down, in which case the remaining server will (over time) reclaim all the addresses the other server had available for allocation, and begin to reuse them. This is called putting the server into the PARTNER-DOWN state.

You can put the server into the PARTNER-DOWN state either by using the **omshell (1)** command or by stopping the server, editing the last peer state declaration in the lease file, and restarting the server. If you use this last method, be sure to leave the date and time of the start of the state blank:

failover peer *name* **state {**
my state partner-down;
peer state *state* **at** *date***;**
}

When the other server comes back online, it should automatically detect that it has been offline and request a complete update from the server that was running in the PARTNER-DOWN state, and then both servers will resume processing together.

It is possible to get into a dangerous situation: if you put one server into the PARTNER-DOWN state, and then *that* server goes down, and the other server comes back up, the other server will not know that the first server was in the PARTNER-DOWN state, and may issue addresses previously issued by the other server to different clients, resulting in IP address conflicts. Before putting a server into PARTNER-DOWN state, therefore, make *sure* that the other server will not restart automatically.

The failover protocol defines a primary server role and a secondary server role. There are some differences in how primaries and secondaries act, but most of the differences simply have to do with providing a way for each peer to behave in the opposite way from the other. So one server must be configured as primary, and the other must be configured as secondary, and it doesn't matter too much which one is which.

FAILOVER STARTUP

When a server starts that has not previously communicated with its failover peer, it must establish communications with its failover peer and synchronize with it before it can serve clients. This can happen either because you have just configured your DHCP servers to perform failover for the first time, or because one of your failover servers has failed catastrophically and lost its database.

The initial recovery process is designed to ensure that when one failover peer loses its database and then resynchronizes, any leases that the failed server gave out before it failed will be honored. When the failed server starts up, it notices that it has no saved failover state, and attempts to contact its peer.

When it has established contact, it asks the peer for a complete copy its peer's lease database. The peer then sends its complete database, and sends a message indicating that it is done. The failed server then waits until MCLT has passed, and once MCLT has passed both servers make the transition back into normal operation. This waiting period ensures that any leases the failed server may have given out while out of contact with its partner will have expired.

While the failed server is recovering, its partner remains in the partner-down state, which means that it is serving all clients. The failed server provides no service at all to DHCP clients until it has made the transition into normal operation.

In the case where both servers detect that they have never before communicated with their partner, they both come up in this recovery state and follow the procedure we have just described. In this case, no service will be provided to DHCP clients until MCLT has expired.

CONFIGURING FAILOVER

In order to configure failover, you need to write a peer declaration that configures the failover protocol, and you need to write peer references in each pool declaration for which you want to do failover. You do not have to do failover for all pools on a given network segment. You must not tell one server it's doing failover on a particular address pool and tell the other it is not. You must not have any common address pools on which you are not doing failover. A pool declaration that uses failover would look like this:

```
pool {
        failover peer "foo";
        deny dynamic bootp clients;
        pool specific parameters
};
```

Dynamic BOOTP leases are not compatible with failover, and, as such, you need to disallow BOOTP in

pools that you are using failover for.

The server currently does very little sanity checking, so if you configure it wrong, it will just fail in odd ways. I would recommend therefore that you either do failover or don't do failover, but don't do any mixed pools. Also, use the same master configuration file for both servers, and have a separate file that contains the peer declaration and includes the master file. This will help you to avoid configuration mismatches. As our implementation evolves, this will become less of a problem. A basic sample dhcpd.conf file for a primary server might look like this:

```
failover peer "foo" {
  primary;
  address anthrax.rc.vix.com;
  port 519;
  peer address trantor.rc.vix.com;
  peer port 520;
  max-response-delay 60;
  max-unacked-updates 10;
  mclt 3600;
  split 128;
  load balance max seconds 3;
}

include "/etc/dhcpd.master";
```

The statements in the peer declaration are as follows:

The *primary* and *secondary* statements

[**primary** | **secondary**]**;**

This determines whether the server is primary or secondary, as described earlier under DHCP FAILOVER.

The *address* statement

address *address***;**

The **address** statement declares the IP address or DNS name on which the server should listen for connections from its failover peer, and also the value to use for the DHCP Failover Protocol server identifier. Because this value is used as an identifier, it may not be omitted.

The *peer address* statement

peer address *address***;**

The **peer address** statement declares the IP address or DNS name to which the server should connect to reach its failover peer for failover messages.

The *port* statement

port *port-number***;**

The **port** statement declares the TCP port on which the server should listen for connections from its failover peer. This statement may not currently be omitted, because the failover protocol does not yet have a reserved TCP port number.

The *peer port* statement

peer port *port-number***;**

The **peer port** statement declares the TCP port to which the server should connect to reach its failover peer for failover messages. This statement may not be omitted because the failover protocol does not yet have a reserved TCP port number. The port number declared in the **peer port** statement may be the same as the port number declared in the **port** statement.

The *max-response-delay* statement

max-response-delay *seconds***;**

The **max-response-delay** statement tells the DHCP server how many seconds may pass without receiving a message from its failover peer before it assumes that connection has failed. This number should be small enough that a transient network failure that breaks the connection will not result in the servers being out of communication for a long time, but large enough that the server isn't constantly making and breaking connections. This parameter must be specified.

The *max-unacked-updates* statement

max-unacked-updates *count***;**

The **max-unacked-updates** statement tells the DHCP server how many BNDUPD messages it can send before it receives a BNDACK from the failover peer. We don't have enough operational experience to say what a good value for this is, but 10 seems to work. This parameter must be specified.

The *mclt* statement

mclt *seconds***;**

The **mclt** statement defines the Maximum Client Lead Time. It must be specified on the primary, and may not be specified on the secondary. This is the length of time for which a lease may be renewed by either failover peer without contacting the other. The longer you set this, the longer it will take for the running server to recover IP addresses after moving into PARTNER-DOWN state. The shorter you set it, the more load your servers will experience when they are not communicating. A value of something like 3600 is probably reasonable, but again bear in mind that we have no real operational experience with this.

The *split* statement

split *index***;**

The split statement specifies the split between the primary and secondary for the purposes of load balancing. Whenever a client makes a DHCP request, the DHCP server runs a hash on the client identification. If the hash comes out to less than the split value, the primary answers. If it comes out to equal to or more than the split, the secondary answers. The only meaningful value is 128, and can only be configured on the primary.

The *hba* statement

hba *colon-separated-hex-list***;**

The hba statement specifies the split between the primary and secondary as a bitmap rather than a cutoff, which theoretically allows for finer-grained control. In practice, there is probably no need for such fine-grained control, however. An example hba statement:

```
hba ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00;
```

This is equivalent to a **split 128;** statement. You must only have **split** or **hba** defined, never both. For most cases, the fine-grained control that **hba** offers isn't necessary, and **split** should be used. As such, the use of **hba** is deprecated.

The *load balance max seconds* statement

load balance max seconds *seconds***;**

This statement allows you to configure a cutoff after which load balancing is disabled. The cutoff is based on the number of seconds since the client sent its first DHCPDISCOVER or DHCPREQUEST message, and only works with clients that correctly implement the *secs* field - fortunately most clients do. We recommend setting this to something like 3 or 5. The effect of this is that if one of the failover peers gets into a state where it is responding to failover messages but not responding to some client requests, the other failover peer will take over its client load automatically as the clients retry.

CLIENT CLASSING

Clients can be separated into classes, and treated differently depending on what class they are in. This separation can be done either with a conditional statement, or with a match statement within the class declaration. It is possible to specify a limit on the total number of clients within a particular class or subclass that may hold leases at one time, and it is possible to specify automatic subclassing based on the contents of the client packet.

To add clients to classes based on conditional evaluation, you can specify a matching expression in the class statement:

```
class "ras-clients" {
  match if substring (option dhcp-client-identifier, 1, 3) = "RAS";
}
```

Note that whether you use matching expressions or add statements (or both) to classify clients, you must always write a class declaration for any class that you use. If there will be no match statement and no in-scope statements for a class, the declaration should look like this:

```
class "ras-clients" {
}
```

SUBCLASSES

In addition to classes, it is possible to declare subclasses. A subclass is a class with the same name as a regular class, but with a specific submatch expression which is hashed for quick matching. This is essentially a speed hack - the main difference between five classes with match expressions and one class with five subclasses is that it will be quicker to find the subclasses. Subclasses work as follows:

```
class "allocation-class-1" {
  match pick-first-value (option dhcp-client-identifier, hardware);
}

class "allocation-class-2" {
  match pick-first-value (option dhcp-client-identifier, hardware);
}

subclass "allocation-class-1" 1:8:0:2b:4c:39:ad;
subclass "allocation-class-2" 1:8:0:2b:a9:cc:e3;
subclass "allocation-class-1" 1:0:0:c4:aa:29:44;

subnet 10.0.0.0 netmask 255.255.255.0 {
  pool {
    allow members of "allocation-class-1";
    range 10.0.0.11 10.0.0.50;
  }
  pool {
    allow members of "allocation-class-2";
    range 10.0.0.51 10.0.0.100;
  }
}
```

The data following the class name in the subclass declaration is a constant value to use in matching the match expression for the class. When class matching is done, the server will evaluate the match expression and then look the result up in the hash table. If it finds a match, the client is considered a member of both the class and the subclass.

Subclasses can be declared with or without scope. In the above example, the sole purpose of the subclass is to allow some clients access to one address pool, while other clients are given access to the other pool, so these subclasses are declared without scopes. If part of the purpose of the subclass were to define different parameter values for some clients, you might want to declare some subclasses with scopes.

In the above example, if you had a single client that needed some configuration parameters, while most didn't, you might write the following subclass declaration for that client:

```
subclass "allocation-class-2" 1:08:00:2b:a1:11:31 {
  option root-path "samsara:/var/diskless/alphapc";
  filename "/tftpboot/netbsd.alphapc-diskless";
}
```

In this example, we've used subclassing as a way to control address allocation on a per-client basis. However, it's also possible to use subclassing in ways that are not specific to clients - for example, to use the value of the vendor-class-identifier option to determine what values to send in the vendor-encapsulated-options option. An example of this is shown under the VENDOR ENCAPSULATED OPTIONS head in the **dhcp-options(5)** manual page.

PER-CLASS LIMITS ON DYNAMIC ADDRESS ALLOCATION

You may specify a limit to the number of clients in a class that can be assigned leases. The effect of this will be to make it difficult for a new client in a class to get an address. Once a class with such a limit has reached its limit, the only way a new client in that class can get a lease is for an existing client to relinquish its lease, either by letting it expire, or by sending a DHCPRELEASE packet. Classes with lease limits are specified as follows:

```
class "limited-1" {
  lease limit 4;
}
```

This will produce a class in which a maximum of four members may hold a lease at one time.

SPAWNING CLASSES

It is possible to declare a *spawning class*. A spawning class is a class that automatically produces subclasses based on what the client sends. The reason that spawning classes were created was to make it possible to create lease-limited classes on the fly. The envisioned application is a cable-modem environment where the ISP wishes to provide clients at a particular site with more than one IP address, but does not wish to provide such clients with their own subnet, nor give them an unlimited number of IP addresses from the network segment to which they are connected.

Many cable modem head-end systems can be configured to add a Relay Agent Information option to DHCP packets when relaying them to the DHCP server. These systems typically add a circuit ID or remote ID option that uniquely identifies the customer site. To take advantage of this, you can write a class declaration as follows:

```
class "customer" {
  spawn with option agent.circuit-id;
  lease limit 4;
}
```

Now whenever a request comes in from a customer site, the circuit ID option will be checked against the class's hash table. If a subclass is found that matches the circuit ID, the client will be classified in that subclass and treated accordingly. If no subclass is found matching the circuit ID, a new one will be created and logged in the **dhcpd.leases** file, and the client will be classified in this new class. Once the client has been classified, it will be treated according to the rules of the class, including, in this case, being subject to the per-site limit of four leases.

The use of the subclass spawning mechanism is not restricted to relay agent options - this particular example is given only because it is a fairly straightforward one.

COMBINING MATCH, MATCH IF AND SPAWN WITH

In some cases, it may be useful to use one expression to assign a client to a particular class, and a second expression to put it into a subclass of that class. This can be done by combining the **match if** and **spawn with** statements, or the **match if** and **match** statements. For example:

```
class "jr-cable-modems" {
  match if option dhcp-vendor-identifier = "jrcm";
```

```
  spawn with option agent.circuit-id;
  lease limit 4;
}

class "dv-dsl-modems" {
  match if option dhcp-vendor-identifier = "dvdsl";
  spawn with option agent.circuit-id;
  lease limit 16;
}
```

This allows you to have two classes that both have the same **spawn with** expression without getting the clients in the two classes confused with each other.

DYNAMIC DNS UPDATES

The DHCP server has the ability to dynamically update the Domain Name System. Within the configuration files, you can define how you want the Domain Name System to be updated. These updates are RFC 2136 compliant so any DNS server supporting RFC 2136 should be able to accept updates from the DHCP server.

Two DNS update schemes are currently implemented, and another is planned. The two that are currently available are the ad-hoc DNS update mode and the interim DHCP-DNS interaction draft update mode. If and when the DHCP-DNS interaction draft and the DHCID draft make it through the IETF standards process, there will be a third mode, which will be the standard DNS update method. The DHCP server must be configured to use one of the two currently-supported methods, or not to do dns updates. This can be done with the *ddns-update-style* configuration parameter.

THE AD-HOC DNS UPDATE SCHEME

The ad-hoc Dynamic DNS update scheme is **now deprecated** and **does not work.** In future releases of the ISC DHCP server, this scheme will not likely be available. The interim scheme works, allows for failover, and should now be used. The following description is left here for informational purposes only.

The ad-hoc Dynamic DNS update scheme implemented in this version of the ISC DHCP server is a prototype design, which does not have much to do with the standard update method that is being standardized in the IETF DHC working group, but rather implements some very basic, yet useful, update capabilities. This mode **does not work** with the *failover protocol* because it does not account for the possibility of two different DHCP servers updating the same set of DNS records.

For the ad-hoc DNS update method, the client's FQDN is derived in two parts. First, the hostname is determined. Then, the domain name is determined, and appended to the hostname.

The DHCP server determines the client's hostname by first looking for a *ddns-hostname* configuration option, and using that if it is present. If no such option is present, the server looks for a valid hostname in the FQDN option sent by the client. If one is found, it is used; otherwise, if the client sent a host-name option, that is used. Otherwise, if there is a host declaration that applies to the client, the name from that declaration will be used. If none of these applies, the server will not have a hostname for the client, and will not be able to do a DNS update.

The domain name is determined based strictly on the server configuration, not on what the client sends. First, if there is a *ddns-domainname* configuration option, it is used. Second, if there is a *domain-name* option configured, that is used. Otherwise, the server will not do the DNS update.

The client's fully-qualified domain name, derived as we have described, is used as the name on which an "A" record will be stored. The A record will contain the IP address that the client was assigned in its lease. If there is already an A record with the same name in the DNS server, no update of either the A or PTR records will occur - this prevents a client from claiming that its hostname is the name of some network server. For example, if you have a fileserver called "fs.sneedville.edu", and the client claims its hostname is "fs", no DNS update will be done for that client, and an error message will be logged.

If the A record update succeeds, a PTR record update for the assigned IP address will be done, pointing to the A record. This update is unconditional - it will be done even if another PTR record of the same name

exists. Since the IP address has been assigned to the DHCP server, this should be safe.

Please note that the current implementation assumes clients only have a single network interface. A client with two network interfaces will see unpredictable behavior. This is considered a bug, and will be fixed in a later release. It may be helpful to enable the *one-lease-per-client* parameter so that roaming clients do not trigger this same behavior.

The DHCP protocol normally involves a four-packet exchange - first the client sends a DHCPDISCOVER message, then the server sends a DHCPOFFER, then the client sends a DHCPREQUEST, then the server sends a DHCPACK. In the current version of the server, the server will do a DNS update after it has received the DHCPREQUEST, and before it has sent the DHCPACK. It only sends the DNS update if it has not sent one for the client's address before, in order to minimize the impact on the DHCP server.

When the client's lease expires, the DHCP server (if it is operating at the time, or when next it operates) will remove the client's A and PTR records from the DNS database. If the client releases its lease by sending a DHCPRELEASE message, the server will likewise remove the A and PTR records.

THE INTERIM DNS UPDATE SCHEME

The interim DNS update scheme operates mostly according to several drafts that are being considered by the IETF and are expected to become standards, but are not yet standards, and may not be standardized exactly as currently proposed. These are:

draft-ietf-dhc-ddns-resolution-??.txt
draft-ietf-dhc-fqdn-option-??.txt
draft-ietf-dnsext-dhcid-rr-??.txt

Because our implementation is slightly different than the standard, we will briefly document the operation of this update style here.

The first point to understand about this style of DNS update is that unlike the ad-hoc style, the DHCP server does not necessarily always update both the A and the PTR records. The FQDN option includes a flag which, when sent by the client, indicates that the client wishes to update its own A record. In that case, the server can be configured either to honor the client's intentions or ignore them. This is done with the statement *allow client-updates;* or the statement *ignore client-updates;*. By default, client updates are allowed.

If the server is configured to allow client updates, then if the client sends a fully-qualified domain name in the FQDN option, the server will use that name the client sent in the FQDN option to update the PTR record. For example, let us say that the client is a visitor from the "radish.org" domain, whose hostname is "jschmoe". The server is for the "example.org" domain. The DHCP client indicates in the FQDN option that its FQDN is "jschmoe.radish.org.". It also indicates that it wants to update its own A record. The DHCP server therefore does not attempt to set up an A record for the client, but does set up a PTR record for the IP address that it assigns the client, pointing at jschmoe.radish.org. Once the DHCP client has an IP address, it can update its own A record, assuming that the "radish.org" DNS server will allow it to do so.

If the server is configured not to allow client updates, or if the client doesn't want to do its own update, the server will simply choose a name for the client from either the fqdn option (if present) or the hostname option (if present). It will use its own domain name for the client, just as in the ad-hoc update scheme. It will then update both the A and PTR record, using the name that it chose for the client. If the client sends a fully-qualified domain name in the fqdn option, the server uses only the leftmost part of the domain name - in the example above, "jschmoe" instead of "jschmoe.radish.org".

Also, if the *use-host-decl-names* configuration option is enabled, then the host declaration's *hostname* will be used in place of the *hostname* option, and the same rules will apply as described above.

The other difference between the ad-hoc scheme and the interim scheme is that with the interim scheme, a method is used that allows more than one DHCP server to update the DNS database without accidentally deleting A records that shouldn't be deleted nor failing to add A records that should be added. The scheme works as follows:

When the DHCP server issues a client a new lease, it creates a text string that is an MD5 hash over the DHCP client's identification (see draft-ietf-dnsext-dhcid-rr-??.txt for details). The update adds an A record with the name the server chose and a TXT record containing the hashed identifier string (hashid). If this

update succeeds, the server is done.

If the update fails because the A record already exists, then the DHCP server attempts to add the A record with the prerequisite that there must be a TXT record in the same name as the new A record, and that TXT record's contents must be equal to hashid. If this update succeeds, then the client has its A record and PTR record. If it fails, then the name the client has been assigned (or requested) is in use, and can't be used by the client. At this point the DHCP server gives up trying to do a DNS update for the client until the client chooses a new name.

The interim DNS update scheme is called interim for two reasons. First, it does not quite follow the drafts. The current versions of the drafts call for a new DHCID RRtype, but this is not yet available. The interim DNS update scheme uses a TXT record instead. Also, the existing ddns-resolution draft calls for the DHCP server to put a DHCID RR on the PTR record, but the *interim* update method does not do this. It is our position that this is not useful, and we are working with the author in hopes of removing it from the next version of the draft, or better understanding why it is considered useful.

In addition to these differences, the server also does not update very aggressively. Because each DNS update involves a round trip to the DNS server, there is a cost associated with doing updates even if they do not actually modify the DNS database. So the DHCP server tracks whether or not it has updated the record in the past (this information is stored on the lease) and does not attempt to update records that it thinks it has already updated.

This can lead to cases where the DHCP server adds a record, and then the record is deleted through some other mechanism, but the server never again updates the DNS because it thinks the data is already there. In this case the data can be removed from the lease through operator intervention, and once this has been done, the DNS will be updated the next time the client renews.

DYNAMIC DNS UPDATE SECURITY

When you set your DNS server up to allow updates from the DHCP server, you may be exposing it to unauthorized updates. To avoid this, you should use TSIG signatures - a method of cryptographically signing updates using a shared secret key. As long as you protect the secrecy of this key, your updates should also be secure. Note, however, that the DHCP protocol itself provides no security, and that clients can therefore provide information to the DHCP server which the DHCP server will then use in its updates, with the constraints described previously.

The DNS server must be configured to allow updates for any zone that the DHCP server will be updating. For example, let us say that clients in the sneedville.edu domain will be assigned addresses on the 10.10.17.0/24 subnet. In that case, you will need a key declaration for the TSIG key you will be using, and also two zone declarations - one for the zone containing A records that will be updates and one for the zone containing PTR records - for ISC BIND, something like this:

```
key DHCP_UPDATER {
  algorithm HMAC-MD5.SIG-ALG.REG.INT;
  secret pRP5FapFoJ95JEL06sv4PQ==;
};

zone "example.org" {
        type master;
        file "example.org.db";
        allow-update { key DHCP_UPDATER; };
};

zone "17.10.10.in-addr.arpa" {
        type master;
        file "10.10.17.db";
        allow-update { key DHCP_UPDATER; };
};
```

You will also have to configure your DHCP server to do updates to these zones. To do so, you need to add

something like this to your dhcpd.conf file:

```
key DHCP_UPDATER {
  algorithm HMAC-MD5.SIG-ALG.REG.INT;
  secret pRP5FapFoJ95JEL06sv4PQ==;
};

zone EXAMPLE.ORG. {
  primary 127.0.0.1;
  key DHCP_UPDATER;
}

zone 17.127.10.in-addr.arpa. {
  primary 127.0.0.1;
  key DHCP_UPDATER;
}
```

The *primary* statement specifies the IP address of the name server whose zone information is to be updated.

Note that the zone declarations have to correspond to authority records in your name server - in the above example, there must be an SOA record for "example.org." and for "17.10.10.in-addr.arpa.". For example, if there were a subdomain "foo.example.org" with no separate SOA, you could not write a zone declaration for "foo.example.org." Also keep in mind that zone names in your DHCP configuration should end in a "."; this is the preferred syntax. If you do not end your zone name in a ".", the DHCP server will figure it out. Also note that in the DHCP configuration, zone names are not encapsulated in quotes where there are in the DNS configuration.

You should choose your own secret key, of course. The ISC BIND 8 and 9 distributions come with a program for generating secret keys called dnssec-keygen. The version that comes with BIND 9 is likely to produce a substantially more random key, so we recommend you use that one even if you are not using BIND 9 as your DNS server. If you are using BIND 9's dnssec-keygen, the above key would be created as follows:

```
dnssec-keygen -a HMAC-MD5 -b 128 -n USER DHCP_UPDATER
```

If you are using the BIND 8 dnskeygen program, the following command will generate a key as seen above:

```
dnskeygen -H 128 -u -c -n DHCP_UPDATER
```

You may wish to enable logging of DNS updates on your DNS server. To do so, you might write a logging statement like the following:

```
logging {
        channel update_debug {
                file "/var/log/update-debug.log";
                severity  debug 3;
                print-category      yes;
                print-severity      yes;
                print-time          yes;
        };
        channel security_info       {
                file        "/var/log/named-auth.info";
                severity  info;
                print-category      yes;
                print-severity      yes;
                print-time          yes;
        };

        category update { update_debug; };
        category security { security_info; };
```

};

You must create the /var/log/named-auth.info and /var/log/update-debug.log files before starting the name server. For more information on configuring ISC BIND, consult the documentation that accompanies it.

REFERENCE: EVENTS

There are three kinds of events that can happen regarding a lease, and it is possible to declare statements that occur when any of these events happen. These events are the commit event, when the server has made a commitment of a certain lease to a client, the release event, when the client has released the server from its commitment, and the expiry event, when the commitment expires.

To declare a set of statements to execute when an event happens, you must use the **on** statement, followed by the name of the event, followed by a series of statements to execute when the event happens, enclosed in braces. Events are used to implement DNS updates, so you should not define your own event handlers if you are using the built-in DNS update mechanism.

The built-in version of the DNS update mechanism is in a text string towards the top of server/dhcpd.c. If you want to use events for things other than DNS updates, and you also want DNS updates, you will have to start out by copying this code into your dhcpd.conf file and modifying it.

REFERENCE: DECLARATIONS

The *include* **statement**

include *"filename"***;**

The *include* statement is used to read in a named file, and process the contents of that file as though it were entered in place of the include statement.

The *shared-network* **statement**

shared-network *name* **{**
[*parameters*]
[*declarations*]
}

The *shared-network* statement is used to inform the DHCP server that some IP subnets actually share the same physical network. Any subnets in a shared network should be declared within a *shared-network* statement. Parameters specified in the *shared-network* statement will be used when booting clients on those subnets unless parameters provided at the subnet or host level override them. If any subnet in a shared network has addresses available for dynamic allocation, those addresses are collected into a common pool for that shared network and assigned to clients as needed. There is no way to distinguish on which subnet of a shared network a client should boot.

Name should be the name of the shared network. This name is used when printing debugging messages, so it should be descriptive for the shared network. The name may have the syntax of a valid domain name (although it will never be used as such), or it may be any arbitrary name, enclosed in quotes.

The *subnet* **statement**

subnet *subnet-number* **netmask** *netmask* **{**
[*parameters*]
[*declarations*]
}

The *subnet* statement is used to provide dhcpd with enough information to tell whether or not an IP address is on that subnet. It may also be used to provide subnet-specific parameters and to specify what addresses may be dynamically allocated to clients booting on that subnet. Such addresses are specified using the *range* declaration.

The *subnet-number* should be an IP address or domain name which resolves to the subnet number of the subnet being described. The *netmask* should be an IP address or domain name which resolves to the subnet mask of the subnet being described. The subnet number, together with the netmask, are sufficient to determine whether any given IP address is on the specified subnet.

Although a netmask must be given with every subnet declaration, it is recommended that if there is any variance in subnet masks at a site, a subnet-mask option statement be used in each subnet declaration to set the desired subnet mask, since any subnet-mask option statement will override the subnet mask declared in the subnet statement.

The *range* **statement**

range [**dynamic-bootp**] *low-address* [*high-address*]**;**

For any subnet on which addresses will be assigned dynamically, there must be at least one *range* statement. The range statement gives the lowest and highest IP addresses in a range. All IP addresses in the range should be in the subnet in which the *range* statement is declared. The *dynamic-bootp* flag may be specified if addresses in the specified range may be dynamically assigned to BOOTP clients as well as DHCP clients. When specifying a single address, *high-address* can be omitted.

The *host* **statement**

host *hostname* {
  [*parameters*]
  [*declarations*]
}

The **host** declaration provides a scope in which to provide configuration information about a specific client, and also provides a way to assign a client a fixed address. The host declaration provides a way for the DHCP server to identify a DHCP or BOOTP client, and also a way to assign the client a static IP address.

If it is desirable to be able to boot a DHCP or BOOTP client on more than one subnet with fixed addresses, more than one address may be specified in the *fixed-address* declaration, or more than one **host** statement may be specified matching the same client.

If client-specific boot parameters must change based on the network to which the client is attached, then multiple **host** declarations should be used. The **host** declarations will only match a client if one of their *fixed-address* statements is viable on the subnet (or shared network) where the client is attached. Conversely, for a **host** declaration to match a client being allocated a dynamic address, it must not have any *fixed-address* statements. You may therefore need a mixture of **host** declarations for any given client...some having *fixed-address* statements, others without.

hostname should be a name identifying the host. If a *hostname* option is not specified for the host, *hostname* is used.

Host declarations are matched to actual DHCP or BOOTP clients by matching the dhcp-client-identifier option specified in the *host* declaration to the one supplied by the client, or, if the *host* declaration or the client does not provide a dhcp-client-identifier option, by matching the *hardware* parameter in the *host* declaration to the network hardware address supplied by the client. BOOTP clients do not normally provide a *dhcp-client-identifier*, so the hardware address must be used for all clients that may boot using the BOOTP protocol.

Please be aware that **only** the *dhcp-client-identifier* option and the hardware address can be used to match a host declaration. For example, it is not possible to match a host declaration to a *host-name* option. This is because the host-name option cannot be guaranteed to be unique for any given client, whereas both the hardware address and *dhcp-client-identifier* option are at least theoretically guaranteed to be unique to a given client.

The *group* **statement**

group {
  [*parameters*]
  [*declarations*]
}

The group statement is used simply to apply one or more parameters to a group of declarations. It can be used to group hosts, shared networks, subnets, or even other groups.

REFERENCE: ALLOW AND DENY

The *allow* and *deny* statements can be used to control the response of the DHCP server to various sorts of requests. The allow and deny keywords actually have different meanings depending on the context. In a pool context, these keywords can be used to set up access lists for address allocation pools. In other contexts, the keywords simply control general server behavior with respect to clients based on scope. In a non-pool context, the *ignore* keyword can be used in place of the *deny* keyword to prevent logging of denied requests.

ALLOW DENY AND IGNORE IN SCOPE

The following usages of allow and deny will work in any scope, although it is not recommended that they be used in pool declarations.

The *unknown-clients* **keyword**

allow unknown-clients;
deny unknown-clients;
ignore unknown-clients;

The **unknown-clients** flag is used to tell dhcpd whether or not to dynamically assign addresses to unknown clients. Dynamic address assignment to unknown clients is **allow**ed by default. An unknown client is simply a client that has no host declaration.

The use of this option is now *deprecated*. If you are trying to restrict access on your network to known clients, you should use **deny unknown-clients;** inside of your address pool, as described under the heading ALLOW AND DENY WITHIN POOL DECLARATIONS.

The *bootp* **keyword**

allow bootp;
deny bootp;
ignore bootp;

The **bootp** flag is used to tell dhcpd whether or not to respond to bootp queries. Bootp queries are **allow**ed by default.

This option does not satisfy the requirement of failover peers for denying dynamic bootp clients. The **deny dynamic bootp clients;** option should be used instead. See the ALLOW AND DENY WITHIN POOL DECLARATIONS section of this man page for more details.

The *booting* **keyword**

allow booting;
deny booting;
ignore booting;

The **booting** flag is used to tell dhcpd whether or not to respond to queries from a particular client. This keyword only has meaning when it appears in a host declaration. By default, booting is **allow**ed, but if it is disabled for a particular client, then that client will not be able to get an address from the DHCP server.

The *duplicates* **keyword**

allow duplicates;
deny duplicates;

Host declarations can match client messages based on the DHCP Client Identifier option or based on the client's network hardware type and MAC address. If the MAC address is used, the host declaration will match any client with that MAC address - even clients with different client identifiers. This doesn't normally happen, but is possible when one computer has more than one operating system installed on it - for example, Microsoft Windows and NetBSD or Linux.

The **duplicates** flag tells the DHCP server that if a request is received from a client that matches the MAC address of a host declaration, any other leases matching that MAC address should be discarded by the server, even if the UID is not the same. This is a violation of the DHCP protocol, but can prevent clients whose client identifiers change regularly from holding many leases at the same time. By default, duplicates

are **allow**ed.

The *declines* **keyword**

allow declines;
deny declines;
ignore declines;

The DHCPDECLINE message is used by DHCP clients to indicate that the lease the server has offered is not valid. When the server receives a DHCPDECLINE for a particular address, it normally abandons that address, assuming that some unauthorized system is using it. Unfortunately, a malicious or buggy client can, using DHCPDECLINE messages, completely exhaust the DHCP server's allocation pool. The server will reclaim these leases, but while the client is running through the pool, it may cause serious thrashing in the DNS, and it will also cause the DHCP server to forget old DHCP client address allocations.

The **declines** flag tells the DHCP server whether or not to honor DHCPDECLINE messages. If it is set to **deny** or **ignore** in a particular scope, the DHCP server will not respond to DHCPDECLINE messages.

The *client-updates* **keyword**

allow client-updates;
deny client-updates;

The **client-updates** flag tells the DHCP server whether or not to honor the client's intention to do its own update of its A record. This is only relevant when doing *interim* DNS updates. See the documentation under the heading THE INTERIM DNS UPDATE SCHEME for details.

ALLOW AND DENY WITHIN POOL DECLARATIONS

The uses of the allow and deny keywords shown in the previous section work pretty much the same way whether the client is sending a DHCPDISCOVER or a DHCPREQUEST message - an address will be allocated to the client (either the old address it's requesting, or a new address) and then that address will be tested to see if it's okay to let the client have it. If the client requested it, and it's not okay, the server will send a DHCPNAK message. Otherwise, the server will simply not respond to the client. If it is okay to give the address to the client, the server will send a DHCPACK message.

The primary motivation behind pool declarations is to have address allocation pools whose allocation policies are different. A client may be denied access to one pool, but allowed access to another pool on the same network segment. In order for this to work, access control has to be done during address allocation, not after address allocation is done.

When a DHCPREQUEST message is processed, address allocation simply consists of looking up the address the client is requesting and seeing if it's still available for the client. If it is, then the DHCP server checks both the address pool permit lists and the relevant in-scope allow and deny statements to see if it's okay to give the lease to the client. In the case of a DHCPDISCOVER message, the allocation process is done as described previously in the ADDRESS ALLOCATION section.

When declaring permit lists for address allocation pools, the following syntaxes are recognized following the allow or deny keywords:

known-clients;

If specified, this statement either allows or prevents allocation from this pool to any client that has a host declaration (i.e., is known). A client is known if it has a host declaration in *any* scope, not just the current scope.

unknown-clients;

If specified, this statement either allows or prevents allocation from this pool to any client that has no host declaration (i.e., is not known).

members of "class**";**

If specified, this statement either allows or prevents allocation from this pool to any client that is a member of the named class.

dynamic bootp clients;

If specified, this statement either allows or prevents allocation from this pool to any bootp client.

authenticated clients;

If specified, this statement either allows or prevents allocation from this pool to any client that has been authenticated using the DHCP authentication protocol. This is not yet supported.

unauthenticated clients;

If specified, this statement either allows or prevents allocation from this pool to any client that has not been authenticated using the DHCP authentication protocol. This is not yet supported.

all clients;

If specified, this statement either allows or prevents allocation from this pool to all clients. This can be used when you want to write a pool declaration for some reason, but hold it in reserve, or when you want to renumber your network quickly, and thus want the server to force all clients that have been allocated addresses from this pool to obtain new addresses immediately when they next renew.

REFERENCE: PARAMETERS

The *always-broadcast* statement

always-broadcast *flag***;**

The DHCP and BOOTP protocols both require DHCP and BOOTP clients to set the broadcast bit in the flags field of the BOOTP message header. Unfortunately, some DHCP and BOOTP clients do not do this, and therefore may not receive responses from the DHCP server. The DHCP server can be made to always broadcast its responses to clients by setting this flag to 'on' for the relevant scope; relevant scopes would be inside a conditional statement, as a parameter for a class, or as a parameter for a host declaration. To avoid creating excess broadcast traffic on your network, we recommend that you restrict the use of this option to as few clients as possible. For example, the Microsoft DHCP client is known not to have this problem, as are the OpenTransport and ISC DHCP clients.

The *always-reply-rfc1048* statement

always-reply-rfc1048 *flag***;**

Some BOOTP clients expect RFC1048-style responses, but do not follow RFC1048 when sending their requests. You can tell that a client is having this problem if it is not getting the options you have configured for it and if you see in the server log the message "(non-rfc1048)" printed with each BOOTREQUEST that is logged.

If you want to send rfc1048 options to such a client, you can set the **always-reply-rfc1048** option in that client's host declaration, and the DHCP server will respond with an RFC-1048-style vendor options field. This flag can be set in any scope, and will affect all clients covered by that scope.

The *authoritative* statement

authoritative;

not authoritative;

The DHCP server will normally assume that the configuration information about a given network segment is not known to be correct and is not authoritative. This is so that if a naive user installs a DHCP server not fully understanding how to configure it, it does not send spurious DHCPNAK messages to clients that have obtained addresses from a legitimate DHCP server on the network.

Network administrators setting up authoritative DHCP servers for their networks should always write **authoritative;** at the top of their configuration file to indicate that the DHCP server *should* send DHCPNAK messages to misconfigured clients. If this is not done, clients will be unable to get a correct IP address after changing subnets until their old lease has expired, which could take quite a long time.

Usually, writing **authoritative;** at the top level of the file should be sufficient. However, if a DHCP

server is to be set up so that it is aware of some networks for which it is authoritative and some networks for which it is not, it may be more appropriate to declare authority on a per-network-segment basis.

Note that the most specific scope for which the concept of authority makes any sense is the physical network segment - either a shared-network statement or a subnet statement that is not contained within a shared-network statement. It is not meaningful to specify that the server is authoritative for some subnets within a shared network, but not authoritative for others, nor is it meaningful to specify that the server is authoritative for some host declarations and not others.

The *boot-unknown-clients* statement

boot-unknown-clients *flag***;**

If the *boot-unknown-clients* statement is present and has a value of *false* or *off*, then clients for which there is no *host* declaration will not be allowed to obtain IP addresses. If this statement is not present or has a value of *true* or *on*, then clients without host declarations will be allowed to obtain IP addresses, as long as those addresses are not restricted by *allow* and *deny* statements within their *pool* declarations.

The *ddns-hostname* statement

ddns-hostname *name***;**

The *name* parameter should be the hostname that will be used in setting up the client's A and PTR records. If no ddns-hostname is specified in scope, then the server will derive the hostname automatically, using an algorithm that varies for each of the different update methods.

The *ddns-domainname* statement

ddns-domainname *name***;**

The *name* parameter should be the domain name that will be appended to the client's hostname to form a fully-qualified domain-name (FQDN).

The *ddns-rev-domainname* statement

ddns-rev-domainname *name***;** The *name* parameter should be the domain name that will be appended to the client's reversed IP address to produce a name for use in the client's PTR record. By default, this is "in-addr.arpa.", but the default can be overridden here.

The reversed IP address to which this domain name is appended is always the IP address of the client, in dotted quad notation, reversed - for example, if the IP address assigned to the client is 10.17.92.74, then the reversed IP address is 74.92.17.10. So a client with that IP address would, by default, be given a PTR record of 10.17.92.74.in-addr.arpa.

The *ddns-update-style* parameter

ddns-update-style *style***;**

The *style* parameter must be one of **ad-hoc**, **interim** or **none**. The *ddns-update-style* statement is only meaningful in the outer scope - it is evaluated once after reading the dhcpd.conf file, rather than each time a client is assigned an IP address, so there is no way to use different DNS update styles for different clients.

The *ddns-updates* **statement**

ddns-updates *flag***;**

The *ddns-updates* parameter controls whether or not the server will attempt to do a DNS update when a lease is confirmed. Set this to *off* if the server should not attempt to do updates within a certain scope. The *ddns-updates* parameter is on by default. To disable DNS updates in all scopes, it is preferable to use the *ddns-update-style* statement, setting the style to *none*.

The *default-lease-time* statement

default-lease-time *time***;**

Time should be the length in seconds that will be assigned to a lease if the client requesting the lease does not ask for a specific expiration time.

The *do-forward-updates* statement

do-forward-updates *flag***;**

The *do-forward-updates* statement instructs the DHCP server as to whether it should attempt to update a DHCP client's A record when the client acquires or renews a lease. This statement has no effect unless DNS updates are enabled and **ddns-update-style** is set to **interim**. Forward updates are enabled by default. If this statement is used to disable forward updates, the DHCP server will never attempt to update the client's A record, and will only ever attempt to update the client's PTR record if the client supplies an FQDN that should be placed in the PTR record using the fqdn option. If forward updates are enabled, the DHCP server will still honor the setting of the **client-updates** flag.

The *dynamic-bootp-lease-cutoff* statement

dynamic-bootp-lease-cutoff *date***;**

The *dynamic-bootp-lease-cutoff* statement sets the ending time for all leases assigned dynamically to BOOTP clients. Because BOOTP clients do not have any way of renewing leases, and don't know that their leases could expire, by default dhcpd assigns infinite leases to all BOOTP clients. However, it may make sense in some situations to set a cutoff date for all BOOTP leases - for example, the end of a school term, or the time at night when a facility is closed and all machines are required to be powered off.

Date should be the date on which all assigned BOOTP leases will end. The date is specified in the form:

W YYYY/MM/DD HH:MM:SS

W is the day of the week expressed as a number from zero (Sunday) to six (Saturday). YYYY is the year, including the century. MM is the month expressed as a number from 1 to 12. DD is the day of the month, counting from 1. HH is the hour, from zero to 23. MM is the minute and SS is the second. The time is always in Coordinated Universal Time (UTC), not local time.

The *dynamic-bootp-lease-length* statement

dynamic-bootp-lease-length *length***;**

The *dynamic-bootp-lease-length* statement is used to set the length of leases dynamically assigned to BOOTP clients. At some sites, it may be possible to assume that a lease is no longer in use if its holder has not used BOOTP or DHCP to get its address within a certain time period. The period is specified in *length* as a number of seconds. If a client reboots using BOOTP during the timeout period, the lease duration is reset to *length*, so a BOOTP client that boots frequently enough will never lose its lease. Needless to say, this parameter should be adjusted with extreme caution.

The *filename* statement

filename "*filename*"**;**

The *filename* statement can be used to specify the name of the initial boot file which is to be loaded by a client. The *filename* should be a filename recognizable to whatever file transfer protocol the client can be expected to use to load the file.

The *fixed-address* declaration

fixed-address address [**,** *address* ...]**;**

The *fixed-address* declaration is used to assign one or more fixed IP addresses to a client. It should only appear in a *host* declaration. If more than one address is supplied, then when the client boots, it will be assigned the address that corresponds to the network on which it is booting. If none of the addresses in the *fixed-address* statement are valid for the network to which the client is connected, that client will not match the *host* declaration containing that *fixed-address* declaration. Each *address* in the *fixed-address* declaration should be either an IP address or a domain name that resolves to one or more

IP addresses.

The *get-lease-hostnames* statement

get-lease-hostnames *flag***;**

The *get-lease-hostnames* statement is used to tell dhcpd whether or not to look up the domain name corresponding to the IP address of each address in the lease pool and use that address for the DHCP *hostname* option. If *flag* is true, then this lookup is done for all addresses in the current scope. By default, or if *flag* is false, no lookups are done.

The *hardware* statement

hardware *hardware-type hardware-address***;**

In order for a BOOTP client to be recognized, its network hardware address must be declared using a *hardware* clause in the *host* statement. *hardware-type* must be the name of a physical hardware interface type. Currently, only the **ethernet** and **token-ring** types are recognized, although support for a **fddi** hardware type (and others) would also be desirable. The *hardware-address* should be a set of hexadecimal octets (numbers from 0 through ff) separated by colons. The *hardware* statement may also be used for DHCP clients.

The *lease-file-name* statement

lease-file-name *name***;**

Name should be the name of the DHCP server's lease file. By default, this is DBDIR/dhcpd.leases. This statement **must** appear in the outer scope of the configuration file - if it appears in some other scope, it will have no effect.

The *local-port* statement

local-port *port***;**

This statement causes the DHCP server to listen for DHCP requests on the UDP port specified in *port*, rather than on port 67.

The *local-address* statement

local-address *address***;**

This statement causes the DHCP server to listen for DHCP requests sent to the specified *address*, rather than requests sent to all addresses. Since serving directly attached DHCP clients implies that the server must respond to requests sent to the all-ones IP address, this option cannot be used if clients are on directly attached networks...it is only realistically useful for a server whose only clients are reached via unicasts, such as via DHCP relay agents.

Note: This statement is only effective if the server was compiled using the USE_SOCKETS #define statement, which is default on a small number of operating systems, and must be explicitly chosen at compile-time for all others. You can be sure if your server is compiled with USE_SOCKETS if you see lines of this format at startup:

Listening on Socket/eth0

Note also that since this bind()s all DHCP sockets to the specified address, that only one address may be supported in a daemon at a given time.

The *log-facility* statement

log-facility *facility***;**

This statement causes the DHCP server to do all of its logging on the specified log facility once the dhcpd.conf file has been read. By default the DHCP server logs to the daemon facility. Possible log facilities include auth, authpriv, cron, daemon, ftp, kern, lpr, mail, mark, news, ntp, security, syslog, user, uucp, and local0 through local7. Not all of these facilities are available on all systems, and there may be other facilities available on other systems.

In addition to setting this value, you may need to modify your *syslog.conf* file to configure logging of

the DHCP server. For example, you might add a line like this:

local7.debug /var/log/dhcpd.log

The syntax of the *syslog.conf* file may be different on some operating systems - consult the *syslog.conf* manual page to be sure. To get syslog to start logging to the new file, you must first create the file with correct ownership and permissions (usually, the same owner and permissions of your /var/log/messages or /usr/adm/messages file should be fine) and send a SIGHUP to syslogd. Some systems support log rollover using a shell script or program called newsyslog or logrotate, and you may be able to configure this as well so that your log file doesn't grow uncontrollably.

Because the *log-facility* setting is controlled by the dhcpd.conf file, log messages printed while parsing the dhcpd.conf file or before parsing it are logged to the default log facility. To prevent this, see the README file included with this distribution, which describes how to change the default log facility. When this parameter is used, the DHCP server prints its startup message a second time after parsing the configuration file, so that the log will be as complete as possible.

The *max-lease-time* statement

max-lease-time *time***;**

Time should be the maximum length in seconds that will be assigned to a lease. The only exception to this is that Dynamic BOOTP lease lengths, which are not specified by the client, are not limited by this maximum.

The *min-lease-time* statement

min-lease-time *time***;**

Time should be the minimum length in seconds that will be assigned to a lease.

The *min-secs* statement

min-secs *seconds***;**

Seconds should be the minimum number of seconds since a client began trying to acquire a new lease before the DHCP server will respond to its request. The number of seconds is based on what the client reports, and the maximum value that the client can report is 255 seconds. Generally, setting this to one will result in the DHCP server not responding to the client's first request, but always responding to its second request.

This can be used to set up a secondary DHCP server which never offers an address to a client until the primary server has been given a chance to do so. If the primary server is down, the client will bind to the secondary server, but otherwise clients should always bind to the primary. Note that this does not, by itself, permit a primary server and a secondary server to share a pool of dynamically-allocatable addresses.

The *next-server* statement

next-server *server-name***;**

The *next-server* statement is used to specify the host address of the server from which the initial boot file (specified in the *filename* statement) is to be loaded. *Server-name* should be a numeric IP address or a domain name.

The *omapi-port* statement

omapi-port *port***;**

The *omapi-port* statement causes the DHCP server to listen for OMAPI connections on the specified port. This statement is required to enable the OMAPI protocol, which is used to examine and modify the state of the DHCP server as it is running.

The *one-lease-per-client* statement

one-lease-per-client *flag***;**

If this flag is enabled, whenever a client sends a DHCPREQUEST for a particular lease, the server will

automatically free any other leases the client holds. This presumes that when the client sends a DHCPREQUEST, it has forgotten any lease not mentioned in the DHCPREQUEST - i.e., the client has only a single network interface *and* it does not remember leases it's holding on networks to which it is not currently attached. Neither of these assumptions are guaranteed or provable, so we urge caution in the use of this statement.

The *pid-file-name* statement

pid-file-name *name***;**

Name should be the name of the DHCP server's process ID file. This is the file in which the DHCP server's process ID is stored when the server starts. By default, this is RUNDIR/dhcpd.pid. Like the lease-file-name statement, this statement must appear in the outer scope of the configuration file.

The *ping-check* statement

ping-check *flag***;**

When the DHCP server is considering dynamically allocating an IP address to a client, it first sends an ICMP Echo request (a *ping*) to the address being assigned. It waits for a second, and if no ICMP Echo response has been heard, it assigns the address. If a response *is* heard, the lease is abandoned, and the server does not respond to the client.

This *ping check* introduces a default one-second delay in responding to DHCPDISCOVER messages, which can be a problem for some clients. The default delay of one second may be configured using the ping-timeout parameter. The ping-check configuration parameter can be used to control checking - if its value is false, no ping check is done.

The *ping-timeout* statement

ping-timeout *seconds***;**

If the DHCP server determined it should send an ICMP echo request (a *ping*) because the ping-check statement is true, ping-timeout allows you to configure how many seconds the DHCP server should wait for an ICMP Echo response to be heard, if no ICMP Echo response has been received before the timeout expires, it assigns the address. If a response *is* heard, the lease is abandoned, and the server does not respond to the client. If no value is set, ping-timeout defaults to 1 second.

The *server-identifier* statement

server-identifier *hostname***;**

The server-identifier statement can be used to define the value that is sent in the DHCP Server Identifier option for a given scope. The value specified **must** be an IP address for the DHCP server, and must be reachable by all clients served by a particular scope.

The use of the server-identifier statement is not recommended - the only reason to use it is to force a value other than the default value to be sent on occasions where the default value would be incorrect. The default value is the first IP address associated with the physical network interface on which the request arrived.

The usual case where the *server-identifier* statement needs to be sent is when a physical interface has more than one IP address, and the one being sent by default isn't appropriate for some or all clients served by that interface. Another common case is when an alias is defined for the purpose of having a consistent IP address for the DHCP server, and it is desired that the clients use this IP address when contacting the server.

Supplying a value for the dhcp-server-identifier option is equivalent to using the server-identifier statement.

The *server-name* statement

server-name *name* **;**

The *server-name* statement can be used to inform the client of the name of the server from which it is booting. *Name* should be the name that will be provided to the client.

The *site-option-space* statement

site-option-space *name* **;**

The *site-option-space* statement can be used to determine from what option space site-local options will be taken. This can be used in much the same way as the *vendor-option-space* statement. Site-local options in DHCP are those options whose numeric codes are greater than 128. These options are intended for site-specific uses, but are frequently used by vendors of embedded hardware that contains DHCP clients. Because site-specific options are allocated on an ad hoc basis, it is quite possible that one vendor's DHCP client might use the same option code that another vendor's client uses, for different purposes. The *site-option-space* option can be used to assign a different set of site-specific options for each such vendor, using conditional evaluation (see **dhcp-eval (5)** for details).

The *stash-agent-options* statement

stash-agent-options *flag***;**

If the *stash-agent-options* parameter is true for a given client, the server will record the relay agent information options sent during the client's initial DHCPREQUEST message when the client was in the SELECTING state and behave as if those options are included in all subsequent DHCPREQUEST messages sent in the RENEWING state. This works around a problem with relay agent information options, which is that they usually not appear in DHCPREQUEST messages sent by the client in the RENEWING state, because such messages are unicast directly to the server and not sent through a relay agent.

The *update-optimization* statement

update-optimization *flag***;**

If the *update-optimization* parameter is false for a given client, the server will attempt a DNS update for that client each time the client renews its lease, rather than only attempting an update when it appears to be necessary. This will allow the DNS to heal from database inconsistencies more easily, but the cost is that the DHCP server must do many more DNS updates. We recommend leaving this option enabled, which is the default. This option only affects the behavior of the interim DNS update scheme, and has no effect on the ad-hoc DNS update scheme. If this parameter is not specified, or is true, the DHCP server will only update when the client information changes, the client gets a different lease, or the client's lease expires.

The *update-static-leases* statement

update-static-leases *flag***;**

The *update-static-leases* flag, if enabled, causes the DHCP server to do DNS updates for clients even if those clients are being assigned their IP address using a *fixed-address* statement - that is, the client is being given a static assignment. This can only work with the *interim* DNS update scheme. It is not recommended because the DHCP server has no way to tell that the update has been done, and therefore will not delete the record when it is not in use. Also, the server must attempt the update each time the client renews its lease, which could have a significant performance impact in environments that place heavy demands on the DHCP server.

The *use-host-decl-names* statement

use-host-decl-names *flag***;**

If the *use-host-decl-names* parameter is true in a given scope, then for every host declaration within that scope, the name provided for the host declaration will be supplied to the client as its hostname. So, for example,

```
group {
  use-host-decl-names on;

  host joe {
    hardware ethernet 08:00:2b:4c:29:32;
```

```
    fixed-address joe.fugue.com;
  }
}
```

is equivalent to

```
host joe {
  hardware ethernet 08:00:2b:4c:29:32;
  fixed-address joe.fugue.com;
  option host-name "joe";
}
```

An *option host-name* statement within a host declaration will override the use of the name in the host declaration.

It should be noted here that most DHCP clients completely ignore the host-name option sent by the DHCP server, and there is no way to configure them not to do this. So you generally have a choice of either not having any hostname to client IP address mapping that the client will recognize, or doing DNS updates. It is beyond the scope of this document to describe how to make this determination.

The *use-lease-addr-for-default-route* statement

use-lease-addr-for-default-route *flag***;**

If the *use-lease-addr-for-default-route* parameter is true in a given scope, then instead of sending the value specified in the routers option (or sending no value at all), the IP address of the lease being assigned is sent to the client. This supposedly causes Win95 machines to ARP for all IP addresses, which can be helpful if your router is configured for proxy ARP. The use of this feature is not recommended, because it won't work for many DHCP clients.

The *vendor-option-space* statement

vendor-option-space *string***;**

The *vendor-option-space* parameter determines from what option space vendor options are taken. The use of this configuration parameter is illustrated in the **dhcp-options(5)** manual page, in the *VENDOR ENCAPSULATED OPTIONS* section.

SETTING PARAMETER VALUES USING EXPRESSIONS

Sometimes it's helpful to be able to set the value of a DHCP server parameter based on some value that the client has sent. To do this, you can use expression evaluation. The **dhcp-eval(5)** manual page describes how to write expressions. To assign the result of an evaluation to an option, define the option as follows:

my-parameter **=** *expression* **;**

For example:

```
ddns-hostname = binary-to-ascii (16, 8, "-",
                                 substring (hardware, 1, 6));
```

REFERENCE: OPTION STATEMENTS

DHCP option statements are documented in the **dhcp-options(5)** manual page.

REFERENCE: EXPRESSIONS

Expressions used in DHCP option statements and elsewhere are documented in the **dhcp-eval(5)** manual page.

SEE ALSO

dhcpd(8), dhcpd.leases(5), dhcp-options(5), dhcp-eval(5), RFC2132, RFC2131.

AUTHOR

dhcpd.conf(5) was written by Ted Lemon under a contract with Vixie Labs. Funding for this project was provided by Internet Systems Consortium. Information about Internet Systems Consortium can be found at

http://www.isc.org.

NAME

dhcpd.leases - DHCP client lease database

DESCRIPTION

The Internet Systems Consortium DHCP Server keeps a persistent database of leases that it has assigned. This database is a free-form ASCII file containing a series of lease declarations. Every time a lease is acquired, renewed or released, its new value is recorded at the end of the lease file. So if more than one declaration appears for a given lease, the last one in the file is the current one.

When dhcpd is first installed, there is no lease database. However, dhcpd requires that a lease database be present before it will start. To make the initial lease database, just create an empty file called /var/db/dhcpd.leases. You can do this with:

```
touch DBDIR/dhcpd.leases
```

In order to prevent the lease database from growing without bound, the file is rewritten from time to time. First, a temporary lease database is created and all known leases are dumped to it. Then, the old lease database is renamed /var/db/dhcpd.leases~. Finally, the newly written lease database is moved into place.

FORMAT

Lease descriptions are stored in a format that is parsed by the same recursive descent parser used to read the **dhcpd.conf(5)** and **dhclient.conf(5)** files. Lease files can contain lease declarations, and also group and subgroup declarations, host declarations and failover state declarations. Group, subgroup and host declarations are used to record objects created using the OMAPI protocol.

The lease file is a log-structured file - whenever a lease changes, the contents of that lease are written to the end of the file. This means that it is entirely possible and quite reasonable for there to be two or more declarations of the same lease in the lease file at the same time. In that case, the instance of that particular lease that appears last in the file is the one that is in effect.

Group, subgroup and host declarations in the lease file are handled in the same manner, except that if any of these objects are deleted, a *rubout* is written to the lease file. This is just the same declaration, with **{ deleted; }** in the scope of the declaration. When the lease file is rewritten, any such rubouts that can be eliminated are eliminated. It is possible to delete a declaration in the **dhcpd.conf** file; in this case, the rubout can never be eliminated from the **dhcpd.leases** file.

THE LEASE DECLARATION

lease *ip-address* **{** *statements...* **}**

Each lease declaration includes the single IP address that has been leased to the client. The statements within the braces define the duration of the lease and to whom it is assigned.

starts *date***;**
ends *date***;**
tstp *date***;**
tsfp *date***;**

The start and end time of a lease are recorded using the **starts** and **ends** statements. The **tstp** statement is specified if the failover protocol is being used, and indicates what time the peer has been told the lease expires. The **tsfp** statement is also specified if the failover protocol is being used, and indicates the lease expiry time that the peer has acknowledged. The *date* is specified as follows:

weekday year/*month*/*day hour*:*minute*:*second*

The weekday is present to make it easy for a human to tell when a lease expires - it's specified as a number from zero to six, with zero being Sunday. The day of week is ignored on input. The year is specified with the century, so it should generally be four digits except for really long leases. The month is specified as a number starting with 1 for January. The day of the month is likewise specified starting with 1. The hour is a number between 0 and 23, the minute a number between 0 and 59, and the second also a number between 0 and 59.

Lease times are specified in Coordinated Universal Time (UTC), not in the local time zone. There is probably nowhere in the world where the times recorded on a lease are always the same as wall clock times. On

most unix machines, you can display the current time in UTC by typing **date -u**.

If a lease will never expire, *date* is **never** instead of an actual date.

hardware *hardware-type mac-address***;**

The hardware statement records the MAC address of the network interface on which the lease will be used. It is specified as a series of hexadecimal octets, separated by colons.

uid *client-identifier***;**

The **uid** statement records the client identifier used by the client to acquire the lease. Clients are not required to send client identifiers, and this statement only appears if the client did in fact send one. Client identifiers are normally an ARP type (1 for ethernet) followed by the MAC address, just like in the **hardware** *statement, but this is not required.*

The client identifier is recorded as a colon-separated hexadecimal list or as a quoted string. If it is recorded as a quoted string and it contains one or more non-printable characters, those characters are represented as octal escapes - a backslash character followed by three octal digits.

client-hostname *hostname* **;**

Most DHCP clients will send their hostname in the *host-name* option. If a client sends its hostname in this way, the hostname is recorded on the lease with a **client-hostname** statement. This is not required by the protocol, however, so many specialized DHCP clients do not send a host-name option.

abandoned;

The **abandoned** statement indicates that the DHCP server has abandoned the lease. In that case, the **abandoned** statement will be used to indicate that the lease should not be reassigned. Please see the **dhcpd.conf(5)** manual page for information about abandoned leases.

binding state *state***; next binding state** *state***;**

The **binding state** statement declares the lease's binding state. When the DHCP server is not configured to use the failover protocol, a lease's binding state will be either **active** or **free**. The failover protocol adds some additional transitional states, as well as the **backup** state, which indicates that the lease is available for allocation by the failover secondary.

The **next binding state** statement indicates what state the lease will move to when the current state expires. The time when the current state expires is specified in the *ends* statement.

option agent.circuit-id *string*; **option agent.remote-id** *string*;

The **option agent.circuit-id** and **option agent.remote-id** statements are used to record the circuit ID and remote ID options send by the relay agent, if the relay agent uses the *relay agent information option*. This allows these options to be used consistently in conditional evaluations even when the client is contacting the server directly rather than through its relay agent.

set *variable* **=** *value***;**

The **set** statement sets the value of a variable on the lease. For general information on variables, see the **dhcp-eval(5)** manual page.

The *ddns-text* **variable**

The *ddns-text* variable is used to record the value of the client's TXT identification record when the interim ddns update style has been used to update the DNS for a particular lease.

The *ddns-fwd-name* **variable**

The *ddns-fwd-name* **variable records the value of the name used in updating the client's A record if a DDNS update has been successfully done by the server. The server may also have used this name to update the client's PTR record.**

The *ddns-client-fqdn* **variable**

If the server is configured to use the interim ddns update style, and is also configured to allow clients to

update their own fqdns, and the client did in fact update its own fqdn, then the *ddns-client-fqdn* variable records the name that the client has indicated it is using. This is the name that the server will have used to update the client's PTR record in this case.

The *ddns-rev-name* **variable**

If the server successfully updates the client's PTR record, this variable will record the name that the DHCP server used for the PTR record. The name to which the PTR record points will be either the *ddns-fwd-name* or the *ddns-client-fqdn*.

on *events* **{** *statements...* **}** The **on** *statement records a list of statements to execute if a certain event occurs. The possible events that can occur for an active lease are* **release** and **expiry**. More than one event can be specified - if so, the events are separated by '|' characters.

THE FAILOVER PEER STATE DECLARATION

The state of any failover peering arrangements is also recorded in the lease file, using the **failover peer** statement:

failover peer *name* **state {**
my state *state* **at** *date***;**
peer state *state* **at** *date***;**
}

The states of the peer named *name* is being recorded. Both the state of the running server (**my state**) and the other failover partner (*peer state*) are recorded. The following states are possible: **unknown-state**, **partner-down**, **normal**, **communications-interrupted**, **resolution-interrupted**, **potential-conflict**, **recover**, **recover-done**, **shutdown**, **paused**, and **startup**. **DBDIR/dhcpd.leases**

SEE ALSO

dhcpd(8), dhcp-options(5), dhcp-eval(5), dhcpd.conf(5), RFC2132, RFC2131.

AUTHOR

dhcpd(8) was written by Ted Lemon under a contract with Vixie Labs. Funding for this project was provided by Internet Systems Consortium. Information about Internet Systems Consortium can be found at: **http://www.isc.org/**

NAME

dirent — directory format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dirent.h>
```

DESCRIPTION

Directories provide a convenient hierarchical method of grouping files while obscuring the underlying details of the storage medium. A directory file is differentiated from a plain file by a flag in its inode(5) entry. It consists of records (directory entries) each of which contains information about a file and a pointer to the file itself. Directory entries may contain other directories as well as plain files; such nested directories are referred to as subdirectories. A hierarchy of directories and files is formed in this manner and is called a file system (or referred to as a file system tree).

Each directory file contains two special directory entries; one is a pointer to the directory itself called dot '.' and the other a pointer to its parent directory called dot-dot '..'. Dot and dot-dot are valid pathnames, however, the system root directory '/', has no parent and dot-dot points to itself like dot.

File system nodes are ordinary directory files on which has been grafted a file system object, such as a physical disk or a partitioned area of such a disk. (See mount(8).)

The directory entry format is defined in the file ⟨sys/dirent.h⟩:

```
/*
 * A directory entry has a struct dirent at the front of it, containing
 * its inode number, the length of the entry, and the length of the name
 * contained in the entry.  These are followed by the name padded to an
 * 8 byte boundary with null bytes.  All names are guaranteed null
 * terminated.  The maximum length of a name in a directory is MAXNAMLEN.
 */

struct dirent {
        ino_t     d_fileno;    /* file number of entry */
        uint16_t d_reclen;     /* length of this record */
        uint8_t  d_type;       /* file type, see below */
        uint8_t  d_namlen;     /* length of string in d_name */
#define MAXNAMLEN       511
        char d_name[MAXNAMLEN + 1]; /* name must be no longer than this */
};

/*
 * File types
 */
#define DT_UNKNOWN     0
#define DT_FIFO                1
#define DT_CHR         2
#define DT_DIR         4
#define DT_BLK         6
#define DT_REG         8
#define DT_LNK         10
#define DT_SOCK                12
#define DT_WHT         14
```

SEE ALSO

`getdents`(2), `fs`(5), `inode`(5)

HISTORY

A dir structure appeared in Version 7 AT&T UNIX. The **`dirent`** structure appeared in NetBSD 1.3.

NAME

disklabel — disk pack label

SYNOPSIS

```
#include <sys/disklabel.h>
```

DESCRIPTION

Each disk or disk pack on a system may contain a disk label which provides detailed information about the geometry of the disk and the partitions into which the disk is divided. It should be initialized when the disk is formatted, and may be changed later with the `disklabel`(8) program. This information is used by the system disk driver and by the bootstrap program to determine how to program the drive and where to find the file systems on the disk partitions. Additional information is used by the file system in order to use the disk most efficiently and to locate important file system information. The description of each partition contains an identifier for the partition type (standard file system, swap area, etc.). The file system updates the in-core copy of the label if it contains incomplete information about the file system.

The label is located in the sector number returned by the `getlabelsector`(3) function, usually sector 0 where it may be found without any information about the disk geometry. It is at an offset (specified by the `getlabeloffset`(3) function) from the beginning of the sector, to allow room for the initial bootstrap. The disk sector containing the label is normally made read-only so that it is not accidentally overwritten by pack-to-pack copies or swap operations; the `DIOCWLABEL` `ioctl`(2), which is done as needed by the `disklabel`(8) program.

A copy of the in-core label for a disk can be obtained with the `DIOCGDINFO` `ioctl`(2); this works with a file descriptor for a block or character (‘‘raw’’) device for any partition of the disk. The in-core copy of the label is set by the `DIOCSDINFO` `ioctl`(2). The offset of a partition cannot generally be changed while it is open, nor can it be made smaller while it is open. One exception is that any change is allowed if no label was found on the disk, and the driver was able to construct only a skeletal label without partition information. Finally, the `DIOCWDINFO` `ioctl`(2) operation sets the in-core label and then updates the on-disk label; there must be an existing label on the disk for this operation to succeed. Thus, the initial label for a disk or disk pack must be installed by writing to the raw disk. All of these operations are normally done using `disklabel`(8).

The format of the disk label, as specified in `sys/disklabel.h`, is

```
/*
 * Disk description table, see disktab(5)
 */
#define _PATH_DISKTAB  "/etc/disktab"

/*
 * Each disk has a label which includes information about the hardware
 * disk geometry, file system partitions, and drive specific information.
 * The location of the label, as well as the number of partitions the
 * label can describe and the number of the "whole disk" (raw)
 * partition are machine dependent.
 */
#include <machine/disklabel.h>

/*
 * The absolute maximum number of disk partitions allowed.
 * This is the maximum value of MAXPARTITIONS for which 'struct disklabel'
 * is ≤ DEV_BSIZE bytes long.  If MAXPARTITIONS is greater than this, beware.
 */
```

```
#define MAXMAXPARTITIONS      22
#if MAXPARTITIONS > MAXMAXPARTITIONS
#warning beware: MAXPARTITIONS bigger than MAXMAXPARTITIONS
#endif


#define DISKMAGIC     ((u_int32_t)0x82564557) /* The disk magic number */

#ifndef _LOCORE
struct disklabel {
        u_int32_t d_magic;              /* the magic number */
        u_int16_t d_type;               /* drive type */
        u_int16_t d_subtype;            /* controller/d_type specific */
        char      d_typename[16];       /* type name, e.g. "eagle" */

        /*
         * d_packname contains the pack identifier and is returned when
         * the disklabel is read off the disk or in-core copy.
         * d_boot0 and d_boot1 are the (optional) names of the
         * primary (block 0) and secondary (block 1-15) bootstraps
         * as found in /usr/mdec.  These are returned when using
         * getdiskbyname(3) to retrieve the values from /etc/disktab.
         */
        union {
                char    un_d_packname[16];      /* pack identifier */
                struct {
                        char *un_d_boot0;       /* primary bootstrap name */
                        char *un_d_boot1;       /* secondary bootstrap name */
                } un_b;
        } d_un;
#define d_packname      d_un.un_d_packname
#define d_boot0                 d_un.un_b.un_d_boot0
#define d_boot1                 d_un.un_b.un_d_boot1

                        /* disk geometry: */
        u_int32_t d_secsize;            /* # of bytes per sector */
        u_int32_t d_nsectors;           /* # of data sectors per track */
        u_int32_t d_ntracks;            /* # of tracks per cylinder */
        u_int32_t d_ncylinders;                 /* # of data cylinders per unit */
        u_int32_t d_secpercyl;          /* # of data sectors per cylinder */
        u_int32_t d_secperunit;                 /* # of data sectors per unit */

        /*
         * Spares (bad sector replacements) below are not counted in
         * d_nsectors or d_secpercyl.  Spare sectors are assumed to
         * be physical sectors which occupy space at the end of each
         * track and/or cylinder.
         */
        u_int16_t d_sparespertrack;     /* # of spare sectors per track */
        u_int16_t d_sparespercyl;       /* # of spare sectors per cylinder */
        /*
         * Alternative cylinders include maintenance, replacement,
```

```
         * configuration description areas, etc.
         */
        u_int32_t d_acylinders;             /* # of alt. cylinders per unit */

                        /* hardware characteristics: */
        /*
         * d_interleave, d_trackskew and d_cylskew describe perturbations
         * in the media format used to compensate for a slow controller.
         * Interleave is physical sector interleave, set up by the
         * formatter or controller when formatting.  When interleaving is
         * in use, logically adjacent sectors are not physically
         * contiguous, but instead are separated by some number of
         * sectors.  It is specified as the ratio of physical sectors
         * traversed per logical sector.  Thus an interleave of 1:1
         * implies contiguous layout, while 2:1 implies that logical
         * sector 0 is separated by one sector from logical sector 1.
         * d_trackskew is the offset of sector 0 on track N relative to
         * sector 0 on track N-1 on the same cylinder.  Finally, d_cylskew
         * is the offset of sector 0 on cylinder N relative to sector 0
         * on cylinder N-1.
         */
        u_int16_t d_rpm;              /* rotational speed */
        u_int16_t d_interleave;               /* hardware sector interleave */
        u_int16_t d_trackskew;        /* sector 0 skew, per track */
        u_int16_t d_cylskew;          /* sector 0 skew, per cylinder */
        u_int32_t d_headswitch;               /* head switch time, usec */
        u_int32_t d_trkseek;          /* track-to-track seek, usec */
        u_int32_t d_flags;            /* generic flags */
#define NDDATA 5
        u_int32_t d_drivedata[NDDATA];        /* drive-type specific information */
#define NSPARE 5
        u_int32_t d_spare[NSPARE];    /* reserved for future use */
        u_int32_t d_magic2;           /* the magic number (again) */
        u_int16_t d_checksum;         /* xor of data incl. partitions */

                        /* file system and partition information: */
        u_int16_t d_npartitions;      /* number of partitions in following */
        u_int32_t d_bbsize;           /* size of boot area at sn0, bytes */
        u_int32_t d_sbsize;           /* max size of fs superblock, bytes */
        struct  partition {           /* the partition table */
                u_int32_t p_size;     /* number of sectors in partition */
                u_int32_t p_offset;   /* starting sector */
                u_int32_t p_fsize;    /* file system basic fragment size */
                u_int8_t p_fstype;    /* file system type, see below */
                u_int8_t p_frag;      /* file system fragments per block */
                union {
                        u_int16_t cpg; /* UFS: FS cylinders per group */
                        u_int16_t sgs; /* LFS: FS segment shift */
                } __partition_u1;
#define p_cpg   __partition_u1.cpg
#define p_sgs   __partition_u1.sgs
        } d_partitions[MAXPARTITIONS];        /* actually may be more */
```

```
};
#else /* _LOCORE */
        /*
         * offsets for asm boot files.
         */
        .set    d_secsize,40
        .set    d_nsectors,44
        .set    d_ntracks,48
        .set    d_ncylinders,52
        .set    d_secpercyl,56
        .set    d_secperunit,60
        .set    d_end_,276              /* size of disk label */
#endif /* _LOCORE */

/* d_type values: */
#define DTYPE_SMD       1       /* SMD, XSMD; VAX hp/up */
#define DTYPE_MSCP      2       /* MSCP */
#define DTYPE_DEC       3       /* other DEC (rk, rl) */
#define DTYPE_SCSI      4       /* SCSI */
#define DTYPE_ESDI      5       /* ESDI interface */
#define DTYPE_ST506     6       /* ST506 etc. */
#define DTYPE_HPIB      7       /* CS/80 on HP-IB */
#define DTYPE_HPFL      8       /* HP Fiber-link */
#define DTYPE_FLOPPY    10      /* floppy */
#define DTYPE_CCD       11      /* concatenated disk device */
#define DTYPE_VND       12      /* vnode pseudo-disk */
#define DTYPE_ATAPI     13      /* ATAPI */
#define DTYPE_RAID      14      /* RAIDframe */
#define DTYPE_LD        15      /* logical disk */
#define DTYPE_JFS2      16      /* IBM JFS2 */
#define DTYPE_CGD       17      /* cryptographic pseudo-disk */
#define DTYPE_VINUM     18      /* vinum volume obsolete */
#define DTYPE_FLASH     19      /* flash memory device */

#ifdef DKTYPENAMES
static const char *const dktypenames[] = {
        "unknown",
        "SMD",
        "MSCP",
        "old DEC",
        "SCSI",
        "ESDI",
        "ST506",
        "HP-IB",
        "HP-FL",
        "type 9",
        "floppy",
        "ccd",
        "vnd",
        "ATAPI",
        "RAID",
        "ld",
```

```
        "jfs",
        "cgd",
        "obsolete vinum",
        "flash",
        NULL
};
#define DKMAXTYPES    (sizeof(dktypenames) / sizeof(dktypenames[0]) - 1)
#endif

/*
 * Filesystem type and version.
 * Used to interpret other file system-specific
 * per-partition information.
 */
#define FS_UNUSED      0      /* unused */
#define FS_SWAP        1      /* swap */
#define FS_V6          2      /* Sixth Edition */
#define FS_V7          3      /* Seventh Edition */
#define FS_SYSV        4      /* System V */
#define FS_V71K        5      /* V7 with 1K blocks (4.1, 2.9) */
#define FS_V8          6      /* Eighth Edition, 4K blocks */
#define FS_BSDFFS      7      /* 4.2BSD fast file system */
#define FS_MSDOS       8      /* MSDOS file system */
#define FS_BSDLFS      9      /* 4.4BSD log-structured file system */
#define FS_OTHER       10     /* in use, but unknown/unsupported */
#define FS_HPFS        11     /* OS/2 high-performance file system */
#define FS_ISO9660     12     /* ISO 9660, normally CD-ROM */
#define FS_BOOT        13     /* partition contains bootstrap */
#define FS_ADOS        14     /* AmigaDOS fast file system */
#define FS_HFS         15     /* Macintosh HFS */
#define FS_FILECORE    16     /* Acorn Filecore Filing System */
#define FS_EX2FS       17     /* Linux Extended 2 file system */
#define FS_NTFS        18     /* Windows/NT file system */
#define FS_RAID        19     /* RAIDframe component */
#define FS_CCD         20     /* concatenated disk component */
#define FS_JFS2        21     /* IBM JFS2 */
#define FS_APPLEUFS    22     /* Apple UFS */
#define FS_VINUM       23     /* Vinum */

#ifdef  FSTYPENAMES
static const char *const fstypenames[] = {
        "unused",
        "swap",
        "Version 6",
        "Version 7",
        "System V",
        "4.1BSD",
        "Eighth Edition",
        "4.2BSD",
        "MSDOS",
        "4.4LFS",
        "unknown",
```

```
        "HPFS",
        "ISO9660",
        "boot",
        "ADOS",
        "HFS",
        "FILECORE",
        "Linux Ext2",
        "NTFS",
        "RAID",
        "ccd",
        "jfs",
        "Apple UFS",
        "obsolete vinum",
        NULL
};
#define FSMAXTYPES    (sizeof(fstypenames) / sizeof(fstypenames[0]) - 1)
#endif

/*
 * flags shared by various drives:
 */
#define D_REMOVABLE    0x01    /* removable media */
#define D_ECC          0x02    /* supports ECC */
#define D_BADSECT      0x04    /* supports bad sector forw. */
#define D_RAMDISK      0x08    /* disk emulator */
#define D_CHAIN        0x10    /* can do back-back transfers */

/*
 * Drive data for SMD.
 */

#define d_smdflags     d_drivedata[0]
#define D_SSE          0x1     /* supports skip sectoring */
#define d_mindist      d_drivedata[1]
#define d_maxdist      d_drivedata[2]
#define d_sdist        d_drivedata[3]

/*
 * Drive data for ST506.
 */
#define d_precompcyl   d_drivedata[0]
#define d_gap3         d_drivedata[1] /* used only when formatting */

/*
 * Drive data for SCSI.
 */
#define d_blind        d_drivedata[0]

#ifndef _LOCORE
/*
 * Structure used to perform a format or other raw operation,
 * returning data and/or register values.  Register identification
```

```
 * and format are device- and driver-dependent.
 */
struct format_op {
        char    *df_buf;
        int     df_count;       /* value-result */
        daddr_t df_startblk;
        int     df_reg[8];      /* result */
};

/*
 * Structure used internally to retrieve information about a partition
 * on a disk.
 */
struct partinfo {
        struct  disklabel *disklab;
        struct  partition *part;
};
```

Disk specific ioctls are defined in `sys/dkio.h`.

```
/*
 * Disk-specific ioctls.
 */
        /* get and set disklabel; DIOCGPART used internally */
#define DIOCGDINFO   _IOR('d', 101, struct disklabel) /* get */
#define DIOCSDINFO   _IOW('d', 102, struct disklabel) /* set */
#define DIOCWDINFO   _IOW('d', 103, struct disklabel) /* set, update disk */
#define DIOCGPART    _IOW('d', 104, struct partinfo)  /* get partition */

        /* do format operation, read or write */
#define DIOCRFORMAT    _IOWR('d', 105, struct format_op)
#define DIOCWFORMAT    _IOWR('d', 106, struct format_op)

#define DIOCSSTEP      _IOW('d', 107, int) /* set step rate */
#define DIOCSRETRIES   _IOW('d', 108, int) /* set # of retries */
#define DIOCKLABEL     _IOW('d', 119, int) /* keep/drop label on close? */
#define DIOCWLABEL     _IOW('d', 109, int) /* write en/disable label */

#define DIOCSBAD       _IOW('d', 110, struct dkbad) /* set kernel dkbad */
#define DIOCEJECT      _IOW('d', 112, int) /* eject removable disk */
#define DIOCLOCK       _IOW('d', 113, int) /* lock/unlock pack */

        /* get default label, clear label */
#define DIOCGDEFLABEL  _IOR('d', 114, struct disklabel)
#define DIOCCLRLABEL   _IO('d', 115)
```

SEE ALSO

`disktab`(5), `disklabel`(8), `dkctl`(8)

NAME

disktab — disk description file

SYNOPSIS

```
#include <disktab.h>
```

DESCRIPTION

disktab is a simple database which describes disk geometries and disk partition characteristics. It is used to initialize the disk label on the disk. The format is patterned after the termcap(5) terminal data base. Entries in **disktab** consist of a number of ‘:’ separated fields. The first entry for each disk gives the names which are known for the disk, separated by ‘|’ characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry.

Name	Type	Description
ty	str	Type of disk (e.g. removable, winchester)
dt	str	Type of controller (e.g. SMD, ESDI, floppy)
ns	num	Number of sectors per track
nt	num	Number of tracks per cylinder
nc	num	Total number of cylinders on the disk
sc	num	Number of sectors per cylinder, ns∗nt default
su	num	Number of sectors per unit, sc∗nc default
se	num	Sector size in bytes, DEV_BSIZE default
sf	bool	Controller supports bad144-style bad sector forwarding
rm	num	Rotation speed, rpm, 3600 default
sk	num	Sector skew per track, default 0
cs	num	Sector skew per cylinder, default 0
hs	num	Headswitch time, usec, default 0
ts	num	One-cylinder seek time, usec, default 0
il	num	Sector interleave (n:1), 1 default
d[0-4]	num	Drive-type-dependent parameters
bs	num	Boot block size, default BBSIZE
sb	num	Superblock size, default SBSIZE
ba	num	Block size for partition ‘a’ (bytes)
bd	num	Block size for partition ‘d’ (bytes)
be	num	Block size for partition ‘e’ (bytes)
bf	num	Block size for partition ‘f’ (bytes)
bg	num	Block size for partition ‘g’ (bytes)
bh	num	Block size for partition ‘h’ (bytes)
fa	num	Fragment size for partition ‘a’ (bytes)
fd	num	Fragment size for partition ‘d’ (bytes)
fe	num	Fragment size for partition ‘e’ (bytes)
ff	num	Fragment size for partition ‘f’ (bytes)
fg	num	Fragment size for partition ‘g’ (bytes)
fh	num	Fragment size for partition ‘h’ (bytes)
oa	num	Offset of partition ‘a’ in sectors
ob	num	Offset of partition ‘b’ in sectors
oc	num	Offset of partition ‘c’ in sectors
od	num	Offset of partition ‘d’ in sectors

oe	num	Offset of partition ‘e’ in sectors
of	num	Offset of partition ‘f’ in sectors
og	num	Offset of partition ‘g’ in sectors
oh	num	Offset of partition ‘h’ in sectors
pa	num	Size of partition ‘a’ in sectors
pb	num	Size of partition ‘b’ in sectors
pc	num	Size of partition ‘c’ in sectors
pd	num	Size of partition ‘d’ in sectors
pe	num	Size of partition ‘e’ in sectors
pf	num	Size of partition ‘f’ in sectors
pg	num	Size of partition ‘g’ in sectors
ph	num	Size of partition ‘h’ in sectors
ta	str	Partition type of partition ‘a’ (4.2BSD filesystem, swap, etc)
tb	str	Partition type of partition ‘b’
tc	str	Partition type of partition ‘c’
td	str	Partition type of partition ‘d’
te	str	Partition type of partition ‘e’
tf	str	Partition type of partition ‘f’
tg	str	Partition type of partition ‘g’
th	str	Partition type of partition ‘h’

FILES

`/etc/disktab`

SEE ALSO

`getdiskbyname`(3), `disklabel`(5), `disklabel`(8), `newfs`(8)

HISTORY

The **`disktab`** description file appeared in 4.2BSD.

NAME

dm.conf — dungeon master configuration file

DESCRIPTION

The **dm.conf** file is the configuration file for the dm(8) program. It consists of lines beginning with one of three keywords, *badtty*, *game*, and *time*. All other lines are ignored.

Any tty listed after the keyword *badtty* may not have games played on it. Entries consist of two white-space separated fields: the string *badtty* and the ttyname as returned by ttyname(3). For example, to keep the uucp dialout, "tty19", from being used for games, the entry would be:

```
badtty  /dev/tty19
```

Any day/hour combination listed after the keyword *time* will disallow games during those hours. Entries consist of four white-space separated fields: the string *time*, the unabbreviated day of the week and the beginning and ending time of a period of the day when games may not be played. The time fields are in a 0 based, 24-hour clock. For example, the following entry allows games playing before 8AM and after 5PM on Mondays:

```
time    Monday  8       17
```

Any game listed after the keyword *game* will set parameters for a specific game. Entries consist of five white-space separated fields: the keyword *game*, the name of a game, the highest system load average at which the game may be played, the maximum users allowed if the game is to be played, and the priority at which the game is to be run. Any of these fields may start with a non-numeric character, resulting in no game limitation or priority based on that field.

The game *default* controls the settings for any game not otherwise listed, and must be the last *game* entry in the file. Priorities may not be negative. For example, the following entries limits the game "hack" to running only when the system has 10 or less users and a load average of 5 or less; all other games may be run any time the system has 15 or less users.

```
game    hack            5       10      *
game    default         *       15      *
```

FILES

`/etc/dm.conf` The dm(8) configuration file.

SEE ALSO

setpriority(2), ttyname(3), dm(8)

NAME

editrc — configuration file for editline library

SYNOPSIS

editrc

DESCRIPTION

The **editrc** file defines various settings to be used by the editline(3) library.

The format of each line is:

```
[prog:]command [arg [...]]
```

command is one of the editline(3) builtin commands. Refer to **BUILTIN COMMANDS** for more information.

prog is the program name string that a program defines when it calls el_init(3) to set up editline(3), which is usually *argv[0]*. *command* will be executed for any program which matches *prog*.

prog may also be a regex(3) style regular expression, in which case *command* will be executed for any program that matches the regular expression.

If *prog* is absent, *command* is executed for all programs.

BUILTIN COMMANDS

The **editline** library has some builtin commands, which affect the way that the line editing and history functions operate. These are based on similar named builtins present in the tcsh(1) shell.

The following builtin commands are available:

bind [**-a**] [**-e**] [**-k**] [**-l**] [**-r**] [**-s**] [**-v**] [*key* [*command*]]

Without options, list all bound keys, and the editor command to which each is bound. If *key* is supplied, show the bindings for *key*. If *key command* is supplied, bind *command* to *key*. Options include:

-e Bind all keys to the standard GNU Emacs-like bindings.

-v Bind all keys to the standard vi(1)-like bindings.

-a List or change key bindings in the vi(1) mode alternate (command mode) key map.

-k *key* is interpreted as a symbolic arrow key name, which may be one of 'up', 'down', 'left' or 'right'.

-l List all editor commands and a short description of each.

-r Remove a key's binding.

-s *command* is taken as a literal string and treated as terminal input when *key* is typed. Bound keys in *command* are themselves reinterpreted, and this continues for ten levels of interpretation.

command may be one of the commands documented in **EDITOR COMMANDS** below, or another key.

key and *command* can contain control characters of the form '^*character*' (e.g. '^A'), and the following backslashed escape sequences:

\a Bell

\b Backspace
\e Escape
\f Formfeed
\n Newline
\r Carriage return
\t Horizontal tab
\v Vertical tab
nnn
The ASCII character corresponding to the octal number *nnn*.

'\' nullifies the special meaning of the following character, if it has any, notably '\' and '^'.

echotc [**-sv**] *arg ...*
Exercise terminal capabilities given in *arg ...*. If *arg* is 'baud', 'cols', 'lines', 'rows', 'meta or' 'tabs', the value of that capability is printed, with "yes" or "no" indicating that the terminal does or does not have that capability.

-s returns an empty string for non-existent capabilities, rather than causing an error. **-v** causes messages to be verbose.

edit [on | off]
Enable or disable the **editline** functionality in a program.

history *list* | *size* n | *unique* n
The *list* command lists all entries in the history. The *size* command sets the history size to n entries. The *unique* command controls if history should keep duplicate entries. If n is non zero, only keep unique history entries. If n is zero, then keep all entries (the default).

telltc
List the values of all the terminal capabilities (see termcap(5)).

settc *cap val*
Set the terminal capability *cap* to *val*, as defined in termcap(5). No sanity checking is done.

setty [**-a**] [**-d**] [**-q**] [**-x**] [+*mode*] [-*mode*] [*mode*] [*char=c*]
Control which tty modes that **editrc** won't allow the user to change. **-d**, **-q** or **-x** tells **setty** to act on the 'edit', 'quote' or 'execute' set of tty modes respectively; defaulting to **-x**.

Without other arguments, **setty** lists the modes in the chosen set which are fixed on ('+mode') or off ('-mode'). **-a** lists all tty modes in the chosen set regardless of the setting. With +*mode*, -*mode* or *mode*, fixes *mode* on or off or removes control of *mode* in the chosen set.

Setty can also be used to set tty characters to particular values using *char=value*. If *value* is empty then the character is set to _POSIX_VDISABLE.

EDITOR COMMANDS

The following editor commands are available for use in key bindings:

vi-paste-next
Vi paste previous deletion to the right of the cursor.

vi-paste-prev
Vi paste previous deletion to the left of the cursor.

vi-prev-space-word
Vi move to the previous space delimited word.

vi-prev-word
Vi move to the previous word.

vi-next-space-word
Vi move to the next space delimited word.

vi-next-word
Vi move to the next word.

vi-change-case
Vi change case of character under the cursor and advance one character.

vi-change-meta
Vi change prefix command.

vi-insert-at-bol
Vi enter insert mode at the beginning of line.

vi-replace-char
Vi replace character under the cursor with the next character typed.

vi-replace-mode
Vi enter replace mode.

vi-substitute-char
Vi replace character under the cursor and enter insert mode.

vi-substitute-line
Vi substitute entire line.

vi-change-to-eol
Vi change to end of line.

vi-insert
Vi enter insert mode.

vi-add
Vi enter insert mode after the cursor.

vi-add-at-eol
Vi enter insert mode at end of line.

vi-delete-meta
Vi delete prefix command.

vi-end-word
Vi move to the end of the current space delimited word.

vi-to-end-word
Vi move to the end of the current word.

vi-undo
Vi undo last change.

vi-command-mode
Vi enter command mode (use alternative key bindings).

vi-zero
Vi move to the beginning of line.

vi-delete-prev-char
Vi move to previous character (backspace).

vi-list-or-eof
Vi list choices for completion or indicate end of file if empty line.

vi-kill-line-prev
Vi cut from beginning of line to cursor.

vi-search-prev
Vi search history previous.

vi-search-next
Vi search history next.

vi-repeat-search-next
Vi repeat current search in the same search direction.

vi-repeat-search-prev
Vi repeat current search in the opposite search direction.

vi-next-char
Vi move to the character specified next.

vi-prev-char
Vi move to the character specified previous.

vi-to-next-char
Vi move up to the character specified next.

vi-to-prev-char
Vi move up to the character specified previous.

vi-repeat-next-char
Vi repeat current character search in the same search direction.

vi-repeat-prev-char
Vi repeat current character search in the opposite search direction.

em-delete-or-list
Delete character under cursor or list completions if at end of line.

em-delete-next-word
Cut from cursor to end of current word.

em-yank
Paste cut buffer at cursor position.

em-kill-line
Cut the entire line and save in cut buffer.

em-kill-region
Cut area between mark and cursor and save in cut buffer.

em-copy-region
Copy area between mark and cursor to cut buffer.

em-gosmacs-transpose
Exchange the two characters before the cursor.

em-next-word
Move next to end of current word.

em-upper-case
Uppercase the characters from cursor to end of current word.

em-capitol-case
Capitalize the characters from cursor to end of current word.

em-lower-case
Lowercase the characters from cursor to end of current word.

em-set-mark
Set the mark at cursor.

em-exchange-mark
Exchange the cursor and mark.

em-universal-argument
Universal argument (argument times 4).

em-meta-next
Add 8th bit to next character typed.

em-toggle-overwrite
Switch from insert to overwrite mode or vice versa.

em-copy-prev-word
Copy current word to cursor.

em-inc-search-next
Emacs incremental next search.

em-inc-search-prev
Emacs incremental reverse search.

ed-end-of-file
Indicate end of file.

ed-insert
Add character to the line.

ed-delete-prev-word
Delete from beginning of current word to cursor.

ed-delete-next-char
Delete character under cursor.

ed-kill-line
Cut to the end of line.

ed-move-to-end
Move cursor to the end of line.

ed-move-to-beg
Move cursor to the beginning of line.

ed-transpose-chars
Exchange the character to the left of the cursor with the one under it.

ed-next-char
Move to the right one character.

ed-prev-word
Move to the beginning of the current word.

ed-prev-char
Move to the left one character.

ed-quoted-insert
Add the next character typed verbatim.

ed-digit
Adds to argument or enters a digit.

ed-argument-digit
Digit that starts argument.

ed-unassigned
Indicates unbound character.

ed-tty-sigint
Tty interrupt character.

ed-tty-dsusp
Tty delayed suspend character.

ed-tty-flush-output
Tty flush output characters.

ed-tty-sigquit
Tty quit character.

ed-tty-sigtstp
Tty suspend character.

ed-tty-stop-output
Tty disallow output characters.

ed-tty-start-output
Tty allow output characters.

ed-newline
Execute command.

ed-delete-prev-char
Delete the character to the left of the cursor.

ed-clear-screen
Clear screen leaving current line at the top.

ed-redisplay
Redisplay everything.

ed-start-over
Erase current line and start from scratch.

ed-sequence-lead-in
First character in a bound sequence.

ed-prev-history
Move to the previous history line.

ed-next-history
Move to the next history line.

ed-search-prev-history
Search previous in history for a line matching the current.

ed-search-next-history
Search next in history for a line matching the current.

ed-prev-line
Move up one line.

ed-next-line
Move down one line.

ed-command
Editline extended command.

SEE ALSO

editline(3), regex(3), termcap(5)

AUTHORS

The **editline** library was written by Christos Zoulas, and this manual was written by Luke Mewburn, with some sections inspired by tcsh(1).

NAME

ELF — executable and linking format

SYNOPSIS

```
#include <elf.h>
```

DESCRIPTION

Because of the flexible nature of ELF, the structures describing it are available both as 32bit and 64bit versions. This document uses the 32bit versions, refer to ⟨elf.h⟩ for the corresponding 64bit versions.

The four main types of an ELF object file are:

executable — A file suitable for execution. It contains the information required for creating a new process image.

relocatable — Contains the necessary information to be run through the link editor ld(1) to create an executable or a shared library.

shared — The shared object contains necessary information which can be used by either the link editor ld(1) at link time or by the dynamic loader ld.elf_so(1) at run time.

core — A file which describes the virtual address space and register state of a process. Core files are typically used in conjunction with debuggers such as gdb(1).

ELF files have a dual nature. The toolchain, including tools such as the as(1) and linker ld(1), treats them as a set of sections described by their section headers. The system loader treats them as a set of segments described by the program headers.

The general format of an ELF file is the following: The file starts with an ELF header. This is followed by a table of program headers (optional for relocatable and shared files). After this come the sections/segments. The file ends with a table of section headers (optional for executable files).

A segment can be considered to consist of several sections. For example, all executable sections are typically packed into one loadable segment which is read-only and executable (see *p_flags* in the program header). This enables the system to map the entire file with just a few operations, one for each loadable segment, instead of doing numerous map operations for each section separately.

Each file is described by the ELF header:

```
typedef struct {
        unsigned char  e_ident[ELF_NIDENT];
        Elf32_Half     e_type;
        Elf32_Half     e_machine;
        Elf32_Word     e_version;
        Elf32_Addr     e_entry;
        Elf32_Off      e_phoff;
        Elf32_Off      e_shoff;
        Elf32_Word     e_flags;
        Elf32_Half     e_ehsize;
        Elf32_Half     e_phentsize;
        Elf32_Half     e_phnum;
        Elf32_Half     e_shentsize;
        Elf32_Half     e_shnum;
        Elf32_Half     e_shstrndx;
} Elf32_Ehdr;
```

e_ident[] The array contains the following information in the indicated locations:

EI_MAG0	The elements ranging from EI_MAG0 to EI_MAG3 contain the ELF magic number: \0177ELF
EI_CLASS	Contains the address size of the binary, either 32 or 64bit.
EI_DATA	byte order
EI_VERSION	Contains the ELF header version. This is currently always set to 1.
EI_OSABI	Contains the operating system ABI identification. Note that even though the definition ELFOSABI_NETBSD exists, NetBSD uses ELFOSABI_SYSV here, since the NetBSD ABI does not deviate from the standard.
EI_ABIVERSION	ABI version.

e_type Contains the file type identification. It can be either ET_REL, ET_EXEC, ET_DYN, or ET_CORE for relocatable, executable, shared, or core, respectively.

e_machine Contains the machine type, e.g. SPARC, Alpha, MIPS, ...

e_entry The program entry point if the file is executable.

e_phoff The position of the program header table in the file or 0 if it doesn't exist.

e_shoff The position of the section header table in the file or 0 if it doesn't exist.

e_flags Contains processor-specific flags. For example, the SPARC port uses this space to specify what kind of memory store ordering is required.

e_ehsize The size of the ELF header.

e_phentsize The size of an entry in the program header table. All entries are the same size.

e_phnum The number of entries in the program header table, or 0 if none exists.

e_shentsize The size of an entry in the section header table. All entries are the same size.

e_shnum The number of entries in the section header table, or 0 if none exists.

e_shstrndx Contains the index number of the section which contains the section name strings.

Each ELF section in turn is described by the section header:

```
typedef struct {
        Elf32_Word      sh_name;
        Elf32_Word      sh_type;
        Elf32_Word      sh_flags;
        Elf32_Addr      sh_addr;
        Elf32_Off       sh_offset;
        Elf32_Word      sh_size;
        Elf32_Word      sh_link;
        Elf32_Word      sh_info;
        Elf32_Word      sh_addralign;
        Elf32_Word      sh_entsize;
} Elf32_Shdr;
```

sh_name Contains an index to the position in the section header string section where the name of the current section can be found.

sh_type Contains the section type indicator. The more important possible values are:

- `SHT_NULL` Section is inactive. The other fields contain undefined values.
- `SHT_PROGBITS` Section contains program information. It can be for example code, data, or debugger information.
- `SHT_SYMTAB` Section contains a symbol table. This section usually contains all the symbols and is intended for the regular link editor `ld`(1).
- `SHT_STRTAB` Section contains a string table.
- `SHT_RELA` Section contains relocation information with an explicit addend.
- `SHT_HASH` Section contains a symbol hash table.
- `SHT_DYNAMIC` Section contains dynamic linking information.
- `SHT_NOTE` Section contains some special information. The format can be e.g. vendor-specific.
- `SHT_NOBITS` Sections contains information similar to `SHT_PROGBITS`, but takes up no space in the file. This can be used for e.g. bss.
- `SHT_REL` Section contains relocation information without an explicit addend.
- `SHT_SHLIB` This section type is reserved but has unspecified semantics.
- `SHT_DYNSYM` Section contains a symbol table. This symbol table is intended for the dynamic linker, and is kept as small as possible to conserve space, since it must be loaded to memory at run time.

sh_flags Contains the section flags, which can have the following values or any combination of them:

- `SHF_WRITE` Section is writable after it has been loaded.
- `SHF_ALLOC` Section will occupy memory at run time.
- `SHF_EXECINSTR` Section contains executable machine instructions.

sh_addr Address to where the section will be loaded, or 0 if this section does not reside in memory at run time.

sh_offset The byte offset from the beginning of the file to the beginning of this section. If the section is of type `SHT_NOBITS`, this field specifies the conceptual placement in the file.

sh_size The size of the section in the file for all types except `SHT_NOBITS`. For that type the value may differ from zero, but the section will still always take up no space from the file.

sh_link Contains an index to the section header table. The interpretation depends on the section type as follows:

- `SHT_REL`, `SHT_RELA` Section index of the associated symbol table.
- `SHT_SYMTAB`, `SHT_DYNSYM` Section index of the associated string table.
- `SHT_HASH` Section index of the symbol table to which the hash table applies.

SHT_DYNAMIC Section index of of the string table by which entries in this section are used.

sh_info Contains extra information. The interpretation depends on the type as follows:

SHT_REL
SHT_RELA Section index of the section to which the relocation information applies.

SHT_SYMTAB
SHT_DYNSYM Contains a value one greater that the last local symbol table index.

sh_addralign Marks the section alignment requirement. If, for example, the section contains a doubleword, the entire section must be doubleword aligned to ensure proper alignment. Only 0 and integral powers of two are allowed. Values 0 and 1 denote that the section has no alignment.

sh_entsize Contains the entry size of an element for sections which are constructed of a table of fixed-size entries. If the section does not hold a table of fixed-size entries, this value is 0.

Every executable object must contain a program header. The program header contains information necessary in constructing a process image.

```
typedef struct {
        Elf32_Word      p_type;
        Elf32_Off       p_offset;
        Elf32_Addr      p_vaddr;
        Elf32_Addr      p_paddr;
        Elf32_Word      p_filesz;
        Elf32_Word      p_memsz;
        Elf32_Word      p_flags;
        Elf32_Word      p_align;
} Elf32_Phdr;
```

p_type Contains the segment type indicator. The possible values are:

PT_NULL Segment is inactive. The other fields contain undefined values.

PT_LOAD Segment is loadable. It is loaded to the address described by *p_vaddr*. If *p_memsz* is greater than *p_filesz*, the memory range from (*p_vaddr* + *p_filesz*) to (*p_vaddr* + *p_memsz*) is zero-filled when the segment is loaded. *p_filesz* can not be greater than *p_memsz*. Segments of this type are sorted in the header table by *p_vaddr* in ascending order.

PT_DYNAMIC Segment contains dynamic linking information.

PT_INTERP Segment contains a null-terminated path name to the interpreter. This segment may be present only once in a file, and it must appear before any loadable segments. This field will most likely contain the ELF dynamic loader: /libexec/ld.elf_so

PT_NOTE Segment contains some special information. Format can be e.g. vendor-specific.

PT_SHLIB This segment type is reserved but has unspecified semantics. Programs which contain a segment of this type do not conform to the ABI, and must indicate this by setting the appropriate ABI in the ELF header EI_OSABI field.

PT_PHDR The values in a program header of this type specify the characteristics of the program header table itself. For example, the *p_vaddr* field specifies the program header table location in memory once the program is loaded. This field

may not occur more than once, may occur only if the program header table is part of the file memory image, and must come before any loadable segments.

p_offset Contains the byte offset from the beginning of the file to the beginning of this segment.

p_vaddr Contains the virtual memory address to which this segment is loaded.

p_paddr Contains the physical address to which this segment is loaded. This value is usually ignored, but may be used while bootstrapping or in embedded systems.

p_filesz Contains the number of bytes this segment occupies in the file image.

p_memsz Contains the number of bytes this segment occupies in the memory image.

p_flags Contains the segment flags, which specify the permissions for the segment after it has been loaded. The following values or any combination of them is acceptable:

PF_R Segment can be read.

PF_W Segment can be written.

PF_X Segment is executable.

p_align Contains the segment alignment. Acceptable values are 0 and 1 for no alignment, and integral powers of two. *p_vaddr* should equal *p_offset* modulo *p_align*.

SEE ALSO

as(1), gdb(1), ld(1), ld.elf_so(1), execve(2), nlist(3), a.out(5), core(5), link(5), stab(5)

HISTORY

The ELF object file format first appeared in AT&T System V UNIX.

NAME

envsys.conf — Configuration file for the envsys framework

SYNOPSIS

envstat [**-S**] [**-c** */etc/envsys.conf*]

DESCRIPTION

The **envsys.conf** file configures all the features provided by the envsys(4) framework. It consists of a series of device and sensor blocks . Each sensor block defines a group of *properties*. The file format is free-form: new line markers and indentation are ignored. Comments start with a '#' sign and extend until the end of line.

A *property* is like a variable assignment. It has a name, which goes to the left of the equal sign, and a value, which goes to the right. The assignment ends with a semicolon. It looks like:

```
name = value;
```

There is no difference between string or integer values when defining them. The value must be surrounded by double quotes if it contains whitespace.

There can be multiple groups of devices and multiple groups of sensors in the configuration file.

A device block consists of one or more sensor blocks and one or more global properties. It has the following syntax:

```
device_name {
        prop = value;
        ...
        sensor0 {
                prop = value;
                ...
        }
        ...
        sensorN {
                prop = value;
                ...
        }
}
...
```

Device names are those shown by the envstat -D command; sensor blocks are named by the index position in which they are shown.

For example, if we have the following output from the envstat(8) command:

```
CPU Temperature:     32.000 degC
 MB Temperature:     37.000 degC
  Vcore Voltage:      1.232 V
   +3.3 Voltage:      3.248 V
     +5 Voltage:      4.992 V
    +12 Voltage:     11.985 V
  CPU FAN Speed:       1250 RPM
```

sensor0 corresponds to the *CPU Temperature* sensor and sensor6 corresponds to the *CPU FAN Speed* sensor.

There is another way that will give you the correct index sensor; the `envstat -x` command will print the raw XML property list. You only have to find the *index* object in the appropriate dictionary. The object will be shown as:

```
<key>index</key>
<string>sensor2</string>
```

Invalid sensors and devices will be detected by the `envstat`(8) parser and will be reported as errors.

The following properties are provided for sensor blocks (please note that not all properties apply to all type of sensors):

critical-capacity = 10;

Sets a critical capacity limit property of 10 percent in a battery sensor. Battery sensors are those that report a percentage from the `envstat`(8) output.

It is possible to find out if the sensor accepts this property by running `envstat -x` and looking if the *want-percentage* object is defined as *true* on its dictionary. For example:

```
<key>want-percentage</key>
<true/>
```

Only a value between 0 and 100 is allowed. When the limit is reached in the sensor, a *user-capacity* event will be sent to the `powerd`(8) daemon (if running) and will execute the block for this event in `/etc/powerd/scripts/sensor_battery`.

If this property is set, its value will be shown in the `envstat`(8) display output with a column named *`CritCap`*.

critical-max = 70C;

Sets a critical max limit property in a sensor. Note that in this example, we are specifying the '`C`' keyword at the end; that means that this will only be valid for *temperature* sensors and that the value is specified as degrees *Celsius*. If degrees Fahrenheit are wanted, just change use the letter *F*, like:

```
critical-max = 140F;
```

To know sensor type, you have to look at the *type* object in the XML property list. Remember: the XML property list has all the information that the application uses to print the values!

Other sensors that are not of *temperature* type must not include the final character for the unit. A dot is allowed in the value, if it corresponds to the range that the sensor is reporting. When the limit has been reached in the sensor, a *critical-over* event will be sent to the `powerd`(8) daemon (if running) and will execute the block for this event in the appropriate `/etc/powerd/scripts/sensor_foo` script (depending on the sensor's type).

Please note that this property cannot be set in battery sensors (those that have the *want-percentage* object in their dictionary). This rule applies for the `critical-min` property too.

If this property is set, its value will be shown in the `envstat`(8) display output with a column named *`CritMax`*.

critical-min = 1.230;

Sets a critical min limit property in a sensor. The rules for *critical-max* and *critical-min* are the same. When the limit has been reached in the sensor, a *critical-under* event will be sent to the `powerd`(8) daemon (if running) and will execute the block for this event in the appropriate `/etc/powerd/scripts/sensor_foo` script (depending on the sensor's type).

If this property is set, its value will be shown in the `envstat`(8) display output with a column named *`CritMin`*.

description = string

Sets a new description in a sensor. You can set this property in all sensors, except that you won't be able to set a description that is currently used for the specified device.

rfact = 56000

Sets a new resistor factor property in a sensor. This property is only allowed in *Voltage* sensors and *only* if the driver has enabled the appropriate flag for the mentioned sensor. The resistor factor may be used to change the behavior of the value returned by the driver.

If a sensor supports this, the *allow-rfact* object appears enabled (true) in the dictionary.

The following properties are available for device blocks:

refresh-timeout = 10s

This property sets the refresh timeout value in a driver, and will be used to refresh data and check for critical conditions any time the timeout is met. The value may be specified in seconds, minutes or hours. To specify the value in seconds, the *s* character must be appended last, if minutes is desired, a *m* and a *h* for hours. For example *10s* for 10 seconds or *1h* for one hour.

FILES

`/etc/envsys.conf` Default configuration file.

SEE ALSO

`proplib`(3), `envstat`(8), `powerd`(8)

HISTORY

The **`envsys.conf`** configuration file first appeared in NetBSD 5.0.

NAME

ethers — Ethernet host name data base

DESCRIPTION

The **ethers** file maps Ethernet MAC addresses to host names. Lines consist of an address and a host name, separated by any number of blanks and/or tab characters. A '#' character indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Each line in **ethers** has the format:

```
ethernet-MAC-address hostname-or-IP
```

Ethernet MAC addresses are expressed as six hexadecimal numbers separated by colons, e.g. "08:00:20:00:5a:bc". The functions described in ethers(3) and ether_aton(3) can read and produce this format.

The traditional use of **ethers** involved using hostnames for the second argument. This may not be suitable for machines that don't have a common MAC address for all interfaces (i.e., just about every non Sun machine). There should be no problem in using an IP address as the second field if you wish to differentiate between different interfaces on a system.

FILES

/etc/ethers The **ethers** file resides in /etc.

SEE ALSO

ethers(3)

HISTORY

The **ethers** file format was adopted from SunOS and appeared in NetBSD 1.0.

BUGS

A name server should be used instead of a static file.

NAME

exports — define remote mount points for NFS mount requests

SYNOPSIS

exports

DESCRIPTION

The **exports** file specifies remote mount points for the NFS mount protocol per the NFS server specification; see *Network File System Protocol Specification RFC 1094, Appendix A* and *NFS: Network File System Version 3 Specification, Appendix I.*

Each line in the file (other than comment lines that begin with a '#') specifies the mount point(s) and export flags within one local server filesystem for one or more hosts. A host may be specified only once for each local filesystem on the server and there may be only one default entry for each server filesystem that applies to all other hosts. The latter exports the filesystem to the "world" and should be used only when the filesystem contains public information.

If you have modified the `/etc/exports` file, send the mountd a SIGHUP to make it re-read the `/etc/exports` file: "kill -HUP 'cat /var/run/mountd.pid'".

In a mount entry, the first field(s) specify the directory path(s) within a server filesystem that can be mounted on by the corresponding client(s). There are two forms of this specification. The first is to list all mount points as absolute directory paths separated by whitespace. The second is to specify the pathname of the root of the filesystem followed by the **-alldirs** flag; this form allows the host(s) to mount at any point within the filesystem, including regular files. Note that the **-alldirs** option should not be used as a security measure to make clients mount only those subdirectories that they should have access to. A client can still access the whole filesystem via individual RPCs if it wanted to, even if just one subdirectory has been mounted. The pathnames must not have any symbolic links in them and should not have any "". or ".." components. Mount points for a filesystem may appear on multiple lines each with different sets of hosts and export options.

The second component of a line specifies how the filesystem is to be exported to the host set. The option flags specify whether the filesystem is exported read-only or read-write and how the client uid is mapped to user credentials on the server.

Export options are specified as follows:

-maproot=*user* The credential of the specified user is used for remote access by root. The credential includes all the groups to which the user is a member on the local machine (see `id`(1)). The user may be specified by name or number.

-maproot=*user*:*group1*:*group2*:*...* The colon separated list is used to specify the precise credential to be used for remote access by root. The elements of the list may be either names or numbers. Note that user: should be used to distinguish a credential containing no groups from a complete credential for that user.

-mapall=*user* or **-mapall**=*user*:*group1*:*group2*:*...* specifies a mapping for all client uids (including root) using the same semantics as **-maproot**.

The option **-r** is a synonym for **-maproot** in an effort to be backward compatible with older export file formats.

In the absence of **-maproot** and **-mapall** options, remote accesses by root will result in using a credential of -2:-2. All other users will be mapped to their remote credential. If a **-maproot** option is given, remote access by root will be mapped to that credential instead of -2:-2. If a **-mapall** option is given, all users (including root) will be mapped to that credential in place of their own.

The **-kerb** option specifies that the Kerberos authentication server should be used to authenticate and map client credentials. This option is currently not implemented.

The **-ro** option specifies that the filesystem should be exported read-only (default read/write). The option **-o** is a synonym for **-ro** in an effort to be backward compatible with older export file formats.

The **-noresvport** option specifies that NFS RPC calls for the filesystem do not have to come from reserved ports. Normally, clients are required to use reserved ports for operations. Using this option decreases the security of your system.

The **-noresvmnt** option specifies that mount RPC requests for the filesystem do not have to come from reserved ports. Normally, clients are required to use reserved ports for mount requests. Using this option decreases the security of your system.

WebNFS exports strictly according to the spec (RFC 2054 and RFC 2055) can be done with the **-public** flag. However, this flag in itself allows r/w access to all files in the filesystem, not requiring reserved ports and not remapping uids. It is only provided to conform to the spec, and should normally not be used. For a WebNFS export, use the **-webnfs** flag, which implies **-public**, **-mapall**=*nobody* and **-ro**.

A **-index**=*file* option can be used to specify a file whose handle will be returned if a directory is looked up using the public filehandle (WebNFS). This is to mimic the behavior of URLs. If no **-index** option is specified, a directory filehandle will be returned as usual. The **-index** option only makes sense in combination with the **-public** or **-webnfs** flags.

Warning: exporting a filesystem both using WebNFS and read/write in the normal way to other hosts should be avoided in an environment that is vulnerable to IP spoofing. WebNFS enables any client to get filehandles to the exported filesystem. Using IP spoofing, a client could then pretend to be a host to which the same filesystem was exported read/write, and use the handle to gain access to that filesystem.

The third component of a line specifies the host set to which the line applies. If no host set is specified, the filesystem is exported to everyone. The set may be specified in three ways. The first way is to list the host name(s) separated by white space. (Standard internet "dot" addresses may be used in place of names.) The second way is to specify a "netgroup" as defined in the netgroup file (see netgroup(5)). A netgroup that contains an item that does have a host entry is treated like an error. The third way is to specify an internet subnetwork using a network and network mask that is defined as the set of all hosts with addresses within the subnetwork. This latter approach requires less overhead within the kernel and is recommended for cases where the export line refers to a large number of clients within an administrative subnet.

The first two cases are specified by simply listing the name(s) separated by whitespace. All names are checked to see if they are "netgroup" names first and are assumed to be hostnames otherwise. Using the full domain specification for a hostname can normally circumvent the problem of a host that has the same name as a netgroup. The third case is specified by the flag **-network**=*netname*[/*prefixlength*] and optionally **-mask**=*netmask*. The netmask may be specified either by attaching a */prefixlength* to the **-network** option, or by using a separate **-mask** option. If the mask is not specified, it will default to the mask for that network class (A, B or C; see inet(4)).

Scoped IPv6 address must carry scope identifier as documented in inet6(4). For example, "fe80::%ne2/10" is used to specify fe80::/10 on ne2 interface.

For example:

```
/usr /usr/local -maproot=0:10 friends
/usr -maproot=daemon grumpy.cis.uoguelph.ca 131.104.48.16
/usr -ro -mapall=nobody
/u -maproot=bin: -network 131.104.48 -mask 255.255.255.0
/a -network 192.168.0/24
/a -network 3ffe:1ce1:1:fe80::/64
```

```
/u2 -maproot=root friends
/u2 -alldirs -kerb -network cis-net -mask cis-mask
```

Given that `/usr`, `/u`, and `/u2` are local filesystem mount points, the above example specifies the following: `/usr` is exported to hosts *friends* where friends is specified in the netgroup file with users mapped to their remote credentials and root mapped to uid 0 and group 10. It is exported read-write and the hosts in "friends" can mount either `/usr` or `/usr/local`. It is exported to *131.104.48.16* and *grumpy.cis.uoguelph.ca* with users mapped to their remote credentials and root mapped to the user and groups associated with "daemon"; it is exported to the rest of the world as read-only with all users mapped to the user and groups associated with "nobody".

`/u` is exported to all hosts on the subnetwork *131.104.48* with root mapped to the uid for "bin" and with no group access.

`/u2` is exported to the hosts in "friends" with root mapped to uid and groups associated with "root"; it is exported to all hosts on network "cis-net" allowing mounts at any directory within /u2 and mapping all uids to credentials for the principal that is authenticated by a Kerberos ticket.

`/a` is exported to the network 192.168.0.0, with a netmask of 255.255.255.0. However, the netmask length in the entry for /a is not specified through a -mask option, but through the /prefix notation.

`/a` is also exported to the IPv6 network 3ffe:1ce1:1:fe80:: address, using the upper 64 bits as the prefix. Note that, unlike with IPv4 network addresses, the specified network address must be complete, and not just contain the upper bits. With IPv6 addresses, the -mask option must not be used.

FILES

`/etc/exports` The default remote mount-point file.

SEE ALSO

`netgroup`(5), `mountd`(8), `nfsd`(8), `showmount`(8)

CAVEATS

Don't re-export NFS-mounted filesystems unless you are sure of the implications. NFS has some assumptions about the characteristics of the file systems being exported, e.g. when timestamps are updated. Re-exporting should work to some extent and can even be useful in some cases, but don't expect it works as well as with local file systems.

BUGS

The export options are tied to the local mount points in the kernel and must be non-contradictory for any exported subdirectory of the local server mount point. It is recommended that all exported directories within the same server filesystem be specified on adjacent lines going down the tree. You cannot specify a hostname that is also the name of a netgroup. Specifying the full domain specification for a hostname can normally circumvent the problem.

NAME

forward — mail forwarding instructions

DESCRIPTION

The **.forward** file contains a list of mail addresses or programs that the user's mail should be redirected to. If the file is not present, then no mail forwarding will be done. Mail may also be forwarded as the standard input to a program by prefixing the line with the normal shell pipe symbol (|). If arguments are to be passed to the command, then the entire line should be enclosed in quotes. For security reasons, the **.forward** file must be owned by the user the mail is being sent to, or by root, and the user's shell must be listed in `/etc/shells`.

For example, if a **.forward** file contained the following lines:

```
nobody@FreeBSD.org
"|/usr/bin/vacation nobody"
```

Mail would be forwarded to ⟨nobody@FreeBSD.org⟩ and to the program `/usr/bin/vacation` with the single argument *nobody*.

If a local user address is prefixed with a backslash character, mail is delivered directly to the user's mail spool file, bypassing further redirection.

For example, if user chris had a **.forward** file containing the following lines:

```
chris@otherhost
\chris
```

One copy of mail would be forwarded to *chris@otherhost* and another copy would be retained as mail for local user chris.

FILES

`$HOME/.forward` The user's forwarding instructions.

SEE ALSO

`aliases`(5), `mailaddr`(7), `sendmail`(8)

NAME

fs, **inode** — format of file system volume

SYNOPSIS

```
#include <sys/param.h>
#include <ufs/ffs/fs.h>
#include <ufs/ufs/inode.h>
```

DESCRIPTION

The files ⟨ufs/ffs/fs.h⟩ and ⟨ufs/ufs/inode.h⟩ declare several structures and define variables and macros which are used to create and manage the underlying format of file system objects on random access devices (disks).

The block size and number of blocks which comprise a file system are parameters of the file system. Sectors beginning at BBLOCK and continuing for BBSIZE are used for a disklabel and for some hardware primary and secondary bootstrapping programs.

The actual file system begins at sector SBLOCK with the *super-block* that is of size SBSIZE. The following structure described the super-block and is from the file ⟨ufs/ffs/fs.h⟩:

```
#define FS_MAGIC 0x011954
struct fs {
        int32_t  fs_firstfield;       /* historic file system linked list, */
        int32_t  fs_unused_1;  /*     used for incore super blocks */
        int32_t   fs_sblkno;    /* addr of super-block in filesys */
        int32_t   fs_cblkno;    /* offset of cyl-block in filesys */
        int32_t   fs_iblkno;    /* offset of inode-blocks in filesys */
        int32_t   fs_dblkno;    /* offset of first data after cg */
        int32_t  fs_old_cgoffset; /* cylinder group offset in cylinder */
        int32_t  fs_old_cgmask;         /* used to calc mod fs_ntrak */
        int32_t  fs_old_time;  /* last time written */
        int32_t  fs_old_size;  /* number of blocks in fs */
        int32_t  fs_old_dsize; /* number of data blocks in fs */
        int32_t  fs_ncg;       /* number of cylinder groups */
        int32_t  fs_bsize;     /* size of basic blocks in fs */
        int32_t  fs_fsize;     /* size of frag blocks in fs */
        int32_t  fs_frag;      /* number of frags in a block in fs */
/* these are configuration parameters */
        int32_t  fs_minfree;   /* minimum percentage of free blocks */
        int32_t  fs_old_rotdelay; /* num of ms for optimal next block */
        int32_t  fs_old_rps;   /* disk revolutions per second */
/* these fields can be computed from the others */
        int32_t  fs_bmask;     /* ''blkoff'' calc of blk offsets */
        int32_t  fs_fmask;     /* ''fragoff'' calc of frag offsets */
        int32_t  fs_bshift;    /* ''lblkno'' calc of logical blkno */
        int32_t  fs_fshift;    /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
        int32_t  fs_maxcontig; /* max number of contiguous blks */
        int32_t  fs_maxbpg;    /* max number of blks per cyl group */
/* these fields can be computed from the others */
        int32_t  fs_fragshift; /* block to frag shift */
        int32_t  fs_fsbtodb;   /* fsbtodb and dbtofsb shift constant */
        int32_t  fs_sbsize;    /* actual size of super block */
```

```
        int32_t  fs_spare1[2]; /* old fs_csmask */
                               /* old fs_csshift */
        int32_t  fs_nindir;    /* value of NINDIR */
        int32_t  fs_inopb;     /* value of INOPB */
        int32_t  fs_old_nspf;  /* value of NSPF */
/* yet another configuration parameter */
        int32_t  fs_optim;     /* optimization preference, see below */
/* these fields are derived from the hardware */
        int32_t  fs_old_npsect;       /* # sectors/track including spares */
        int32_t  fs_old_interleave;   /* hardware sector interleave */
        int32_t  fs_old_trackskew;    /* sector 0 skew, per track */
/* fs_id takes the space of unused fs_headswitch and fs_trkseek fields */
        int32_t  fs_id[2];     /* unique file system id */
/* sizes determined by number of cylinder groups and their sizes */
        int32_t  fs_old_csaddr;       /* blk addr of cyl grp summary area */
        int32_t  fs_cssize;    /* size of cyl grp summary area */
        int32_t  fs_cgsize;    /* cylinder group size */
/* these fields are derived from the hardware */
        int32_t  fs_spare2;    /* old fs_ntrak */
        int32_t  fs_old_nsect; /* sectors per track */
        int32_t  fs_old_spc;   /* sectors per cylinder */
        int32_t  fs_old_ncyl;  /* cylinders in file system */
        int32_t  fs_old_cpg;   /* cylinders per group */
        int32_t  fs_ipg;       /* inodes per group */
        int32_t  fs_fpg;       /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
        struct  csum fs_old_cstotal;  /* cylinder summary information */
/* these fields are cleared at mount time */
        int8_t   fs_fmod;      /* super block modified flag */
        int8_t   fs_clean;     /* file system is clean flag */
        int8_t   fs_ronly;     /* mounted read-only flag */
        uint8_t  fs_old_flags; /* see FS_ flags below */
        u_char   fs_fsmnt[MAXMNTLEN];  /* name mounted on */
        u_char   fs_volname[MAXVOLLEN];      /* volume name */
        uint64_t fs_swuid;     /* system-wide uid */
        int32_t  fs_pad;
/* these fields retain the current block allocation info */
        int32_t  fs_cgrotor;   /* last cg searched (UNUSED) */
        void    *fs_ocsp[NOCSPTRS];/* padding; was list of fs_cs buffers */
        uint8_t *fs_contigdirs;   /* # of contiguously allocated dirs */
        struct csum *fs_csp;   /* cg summary info buffer for fs_cs */
        int32_t *fs_maxcluster;/* max cluster in each cyl group */
        u_char  *fs_active;    /* used by snapshots to track fs */
        int32_t  fs_old_cpc;   /* cyl per cycle in postbl */
/* this area is otherwise allocated unless fs_old_flags & FS_FLAGS_UPDATED */
        int32_t  fs_maxbsize;  /* maximum blocking factor permitted */
        int64_t  fs_sparecon64[17]; /* old rotation block list head */
        int64_t  fs_sblockloc; /* byte offset of standard superblock */
        struct  csum_total fs_cstotal; /* cylinder summary information */
        int64_t  fs_time;      /* last time written */
        int64_t  fs_size;      /* number of blocks in fs */
        int64_t  fs_dsize;     /* number of data blocks in fs */
```

```
        int64_t  fs_csaddr;    /* blk addr of cyl grp summary area */
        int64_t  fs_pendingblocks; /* blocks in process of being freed */
        int32_t  fs_pendinginodes; /* inodes in process of being freed */
        int32_t  fs_snapinum[FSMAXSNAP]; /* list of snapshot inode numbers */
/* back to stuff that has been around a while */
        int32_t  fs_avgfilesize; /* expected average file size */
        int32_t  fs_avgfpdir;   /* expected # of files per directory */
        int32_t  fs_save_cgsize; /* save real cg size to use fs_bsize */
        int32_t  fs_sparecon32[26]; /* reserved for future constants */
        uint32_t fs_flags;         /* see FS_ flags below */
/* back to stuff that has been around a while (again) */
        int32_t  fs_contigsumsize; /* size of cluster summary array */
        int32_t  fs_maxsymlinklen; /* max length of an internal symlink */
        int32_t  fs_old_inodefmt;  /* format of on-disk inodes */
        uint64_t fs_maxfilesize;  /* maximum representable file size */
        int64_t  fs_qbmask;       /* ~fs_bmask for use with 64-bit size */
        int64_t  fs_qfmask;       /* ~fs_fmask for use with 64-bit size */
        int32_t  fs_state;        /* validate fs_clean field (UNUSED) */
        int32_t  fs_old_postblformat; /* format of positional layout tables */
        int32_t  fs_old_nrpos; /* number of rotational positions */
        int32_t  fs_spare5[2]; /* old fs_postbloff */
                               /* old fs_rotbloff */
        int32_t  fs_magic;     /* magic number */
};
```

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of ‘blocks’. File system blocks of at most size `MAXBSIZE` can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be `DEV_BSIZE`, or some multiple of a `DEV_BSIZE` unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the **blksize**(*fs*, *ip*, *lbn*) macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 can’t be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it).

The *fs_minfree* element gives the minimum acceptable percentage of file system blocks that may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. The *fs_minfree* element may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of *fs_minfree* is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 8, thus the default fragment size is an eighth of the block size.

The element *fs_optim* specifies whether the file system should try to minimize the time spent allocating blocks, or if it should attempt to minimize the space fragmentation on the disk. If the value of fs_minfree (see above) is less than 10%, then the file system defaults to optimizing for space to avoid running out of full sized blocks. If the value of minfree is greater than or equal to 10%, fragmentation is unlikely to be problematical, and the file system defaults to optimizing for time.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. With the default of 8 distinguished rotational positions, the resolution of the summary information is 2ms for a typical 3600 rpm drive.

The element *fs_rotdelay* gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for *fs_rotdelay* is 2ms.

Each file system has a statically allocated number of inodes, determined by its size and the desired number of file data bytes per inode at the time it was created. See newfs(8) for details on how to set this (and other) filesystem parameters. By default, the inode allocation strategy is extremely conservative.

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to (*struct cg*) must keep its size within MINBSIZE. Note that super-blocks are never more than size SBSIZE.

The path name on which the file system is mounted is maintained in *fs_fsmnt*. MAXMNTLEN defines the amount of space allocated in the super-block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. For a 4096 byte block size, it is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from *fs_csaddr* (size *fs_cssize*) in addition to the super-block.

N.B.: **sizeof**(*struct csum*) must be a power of two in order for the **fs_cs**() macro to work.

The *Super-block for a file system*: The size of the rotational layout tables is limited by the fact that the super-block is of size SBSIZE. The size of these tables is *inversely* proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (*fs_cpc*). The size of the rotational layout tables is derived from the number of bytes remaining in (*struct fs*).

The number of blocks of data per cylinder group is limited because cylinder groups are at most one block. The inode and free block tables must fit into a single block after deducting space for the cylinder group structure (*struct cg*).

The *Inode*: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is 'named' by its device/i-number pair. For further information, see the include file ⟨ufs/ufs/inode.h⟩.

SEE ALSO

newfs(8)

HISTORY

A super-block structure named filsys appeared in Version 6 AT&T UNIX. The file system described in this manual appeared in 4.2BSD.

NAME

fstab — filesystem table for devices, types, and mount points

SYNOPSIS

```
#include <fstab.h>
```

DESCRIPTION

The file **fstab** contains descriptive information about the various file systems. **fstab** is only read by programs, and not written; it is the duty of the system administrator to properly create and maintain this file. Each filesystem is described on a separate line; fields on each line are separated by tabs or spaces. Lines beginning with "#" are comments. The order of records in **fstab** is important because fsck(8), mount(8), and umount(8) sequentially iterate through **fstab** doing their respective tasks.

Each configuration line/record in **fstab** has the format:

```
fs_spec fs_file fs_vfstype fs_mntops fs_freq fs_passno
```

The first field, (*fs_spec*), describes the block special device or remote filesystem to be mounted. For filesystems of type *ffs*, the special file name is the block special file name, and not the character special file name. If a program needs the character special file name, the program must create it by appending a "r" after the last "/" in the special file name.

The second field, (*fs_file*), describes the mount point for the filesystem. For swap and dump partitions, this field should be specified as "none".

The third field, (*fs_vfstype*), describes the type of the filesystem. The system currently supports these filesystems:

adosfs	an AmigaDOS filesystem.
cd9660	an ISO 9660 CD-ROM filesystem.
ext2fs	an implementation of the Linux "Second Extended File-system".
fdesc	an implementation of /dev/fd.
ffs	a local UNIX filesystem.
filecore	a filesystem for RISC OS.
kernfs	various and sundry kernel statistics.
lfs	a log-structured file-system.
mfs	a local memory-based UNIX filesystem.
msdos	an MS-DOS "FAT filesystem".
nfs	a Sun Microsystems compatible "Network File System".
ntfs	a filesystem used by Windows NT. Still experimental.
null	a loop-back filesystem, allowing parts of the system to be viewed elsewhere.
overlay	a demonstration of layered filesystems.
portal	a general filesystem interface, currently supports TCP and FS mounts.
procfs	a local filesystem of process information.
ptyfs	a pseudo-terminal device file system.

smbfs a shared resource from an SMB/CIFS file server.

swap a disk partition to be used for swapping and paging.

tmpfs an efficient memory file system.

umap a user and group re-mapping filesystem.

union a translucent filesystem.

The fourth field, (*fs_mntops*), describes the mount options associated with the filesystem. It is formatted as a comma separated list of options. It contains at least the type of mount (see *fs_type* below) plus any additional options appropriate to the filesystem type.

The option "auto" can be used in the "noauto" form to cause a file system not to be mounted automatically (with "mount -a" , or system boot time).

If the options "userquota" and/or "groupquota" are specified, the filesystem is automatically processed by the quotacheck(8) command, and user and/or group disk quotas are enabled with quotaon(8). By default, filesystem quotas are maintained in files named quota.user and quota.group which are located at the root of the associated filesystem. These defaults may be overridden by putting an equal sign and an alternative absolute pathname following the quota option. Thus, if the user quota file for /tmp is stored in /var/quotas/tmp.user, this location can be specified as:

```
userquota=/var/quotas/tmp.user
```

The type of the mount is extracted from the *fs_mntops* field and stored separately in the *fs_type* field (it is not deleted from the *fs_mntops* field). If *fs_type* is "rw" or "ro" then the filesystem whose name is given in the *fs_file* field is normally mounted read-write or read-only on the specified special file. If *fs_type* is "sw" or "dp" then the special file is made available as a piece of swap or dump space by the swapctl(8) command towards the beginning of the system reboot procedure. See swapctl(8) for more information on configuring swap and dump devices. The fields other than *fs_spec* and *fs_type* are unused. If *fs_type* is specified as "xx" the entry is ignored. This is useful to show disk partitions which are currently unused.

The fifth field, (*fs_freq*), is used for these filesystems by the dump(8) command to determine which filesystems need to be dumped. If the fifth field is not present, a value of zero is returned and dump(8) will assume that the filesystem does not need to be dumped.

The sixth field, (*fs_passno*), is used by the fsck(8) program to determine the order in which filesystem checks are done at reboot time. The root filesystem should be specified with a *fs_passno* of 1, and other filesystems should have a *fs_passno* of 2. Filesystems within a drive will be checked sequentially, but filesystems on different drives will be checked at the same time to use parallelism available in the hardware. If the sixth field is not present or zero, a value of zero is returned and fsck(8) will assume that the filesystem does not need to be checked.

```
#define FSTAB_RW        "rw"    /* read-write device */
#define FSTAB_RQ        "rq"    /* read/write with quotas */
#define FSTAB_RO        "ro"    /* read-only device */
#define FSTAB_SW        "sw"    /* swap device */
#define FSTAB_DP        "dp"    /* dump device */
#define FSTAB_XX        "xx"    /* ignore totally */

struct fstab {
        char    *fs_spec;       /* block special device name */
        char    *fs_file;       /* filesystem path prefix */
        char    *fs_vfstype;    /* type of filesystem */
        char    *fs_mntops;     /* comma separated mount options */
```

```
        char    *fs_type;       /* rw, ro, sw, or xx */
        int     fs_freq;        /* dump frequency, in days */
        int     fs_passno;      /* pass number on parallel fsck */
};
```

The proper way to read records from `fstab` is to use the routines `getfsent`(3), `getfsspec`(3), and `getfsfile`(3).

FILES

`/etc/fstab` The location of **`fstab`** configuration file.

`/usr/share/examples/fstab/`
Some useful configuration examples.

SEE ALSO

`getfsent`(3), `mount`(8), `swapctl`(8)

HISTORY

The **`fstab`** file format appeared in 4.0 BSD.

NAME

ftpd.conf — ftpd(8) configuration file

DESCRIPTION

The **ftpd.conf** file specifies various configuration options for ftpd(8) that apply once a user has authenticated their connection.

ftpd.conf consists of a series of lines, each of which may contain a configuration directive, a comment, or a blank line. Directives that appear later in the file override settings by previous directives. This allows 'wildcard' entries to define defaults, and then have class-specific overrides.

A directive line has the format:

```
command class [arguments]
```

A "\" is the escape character; it can be used to escape the meaning of the comment character, or if it is the last character on a line, extends a configuration directive across multiple lines. A "#" is the comment character, and all characters from it to the end of line are ignored (unless it is escaped with the escape character).

Each authenticated user is a member of a *class*, which is determined by ftpusers(5). *class* is used to determine which **ftpd.conf** entries apply to the user. The following special classes exist when parsing entries in **ftpd.conf**:

all Matches any class.
none Matches no class.

Each class has a type, which may be one of:

GUEST Guests (as per the "anonymous" and "ftp" logins). A chroot(2) is performed after login.

CHROOT
chroot(2)ed users (as per ftpchroot(5)). A chroot(2) is performed after login.

REAL Normal users.

The ftpd(8) **STAT** command will return the class settings for the current user as defined by **ftpd.conf**, unless the **private** directive is set for the class.

Each configuration line may be one of:

advertize *class* [*host*]
Set the address to advertise in the response to the **PASV** and **LPSV** commands to the address for *host* (which may be either a host name or IP address). This may be useful in some firewall configurations, although many ftp clients may not work if the address being advertised is different to the address that they've connected to. If *class* is "none" or *host* not is specified, disable this.

checkportcmd *class* [**off**]
Check the **PORT** command for validity. The **PORT** command will fail if the IP address specified does not match the FTP command connection, or if the remote TCP port number is less than IPPORT_RESERVED. It is *strongly* encouraged that this option be used, especially for sites concerned with potential security problems with FTP bounce attacks. If *class* is "none" or **off** is specified, disable this feature, otherwise enable it.

chroot *class* [**pathformat**]
If *pathformat* is not specified or *class* is "none", use the default behavior (see below). Otherwise, *pathformat* is parsed to create a directory to create as the root directory with chroot(2) into upon login.

pathformat can contain the following escape strings:

Escape	Description
%c	Class name.
%d	Home directory of user.
%u	User name.
%%	A "%" character.

The default root directory is:

CHROOT
: The user's home directory.

GUEST
: If **-a** *anondir* is specified, use *anondir*, otherwise the home directory of the 'ftp' user.

REAL
: By default no chroot(2) is performed.

classtype *class type*
: Set the class type of *class* to *type* (see above).

conversion *class suffix* [*type disable command*]
: Define an automatic in-line file conversion. If a file to retrieve ends in *suffix*, and a real file (sans *suffix*) exists, then the output of *command* is returned instead of the contents of the file.

suffix
: The suffix to initiate the conversion.

type
: A list of valid file types for the conversion. Valid types are: 'f' (file), and 'd' (directory).

disable
: The name of file that will prevent conversion if it exists. A file name of "." will prevent this disabling action (i.e., the conversion is always permitted.)

command
: The command to run for the conversion. The first word should be the full path name of the command, as execv(3) is used to execute the command. All instances of the word "%s" in *command* are replaced with the requested file (sans *suffix*).

Conversion directives specified later in the file override earlier conversions with the same suffix.

denyquick *class* [**off**]
: Enforce ftpusers(5) rules after the **USER** command is received, rather than after the **PASS** command is received. Whilst enabling this feature may allow information leakage about available accounts (for example, if you allow some users of a **REAL** or **CHROOT** class but not others), it is useful in preventing a denied user (such as 'root') from entering their password across an insecure connection. This option is *strongly* recommended for servers which run an anonymous-only service. If *class* is "none" or **off** is specified, disable this feature, otherwise enable it.

display *class* [*file*]
: If *file* is not specified or *class* is "none", disable this. Otherwise, each time the user enters a new directory, check if *file* exists, and if so, display its contents to the user. Escape sequences are supported; refer to **Display file escape sequences** in ftpd(8) for more information.

hidesymlinks *class* [**off**]
: If *class* is "none" or **off** is specified, disable this feature. Otherwise, the **LIST** command lists symbolic links as the file or directory the link references ("ls -LlA"). Servers which run an anonymous service may wish to enable this feature for **GUEST** users, so that symbolic links do not leak names in directories that are not searchable by **GUEST** users.

homedir *class* [**pathformat**]
: If *pathformat* is not specified or *class* is "none", use the default behavior (see below). Otherwise, *pathformat* is parsed to create a directory to change into upon login, and to use as the 'home' directory of the user for tilde expansion in pathnames, etc. *pathformat* is parsed as per the **chroot** directive.

The default home directory is the home directory of the user for **REAL** users, and `/` for **GUEST** and **CHROOT** users.

limit *class* [*count* [*file*]]
Limit the maximum number of concurrent connections for *class* to *count*, with '−1' meaning unlimited connections. If the limit is exceeded and *file* is specified, display its contents to the user. If *class* is "none" or *count* is not specified, disable this. If *file* is a relative path, it will be searched for in `/etc` (which can be overridden with **-c** *confdir*).

maxfilesize *class* [*size*]
Set the maximum size of an uploaded file to *size*, with '−1' meaning unlimited connections. If *class* is "none" or *size* is not specified, disable this.

maxtimeout *class* [*time*]
Set the maximum timeout period that a client may request, defaulting to two hours. This cannot be less than 30 seconds, or the value for **timeout**. If *class* is "none" or *time* is not specified, use the default.

mmapsize *class* [*size*]
Set the size of the sliding window to map a file using `mmap`(2). If zero, `ftpd`(8) will use `read`(2) instead. The default is zero. This option affects only binary transfers. If *class* is "none" or *size* is not specified, use the default.

modify *class* [**off**]
If *class* is "none" or **off** is specified, disable the following commands: **CHMOD**, **DELE**, **MKD**, **RMD**, **RNFR**, and **UMASK**. Otherwise, enable them.

motd *class* [*file*]
If *file* is not specified or *class* is "none", disable this. Otherwise, use *file* as the message of the day file to display after login. Escape sequences are supported; refer to **Display file escape sequences** in `ftpd`(8) for more information. If *file* is a relative path, it will be searched for in `/etc` (which can be overridden with **-c** *confdir*).

notify *class* [*fileglob*]
If *fileglob* is not specified or *class* is "none", disable this. Otherwise, each time the user enters a new directory, notify the user of any files matching *fileglob*.

passive *class* [**off**]
If *class* is "none" or **off** is specified, prevent passive (**PASV**, **LPSV**, and **EPSV**) connections. Otherwise, enable them.

portrange *class* [*min max*]
Set the range of port number which will be used for the passive data port. *max* must be greater than *min*, and both numbers must be be between `IPPORT_RESERVED` (1024) and 65535. If *class* is "none" or no arguments are specified, disable this.

private *class* [**off**]
If *class* is "none" or **off** is specified, do not display class information in the output of the **STAT** command. Otherwise, display the information.

rateget *class* [*rate*]
Set the maximum get (**RETR**) transfer rate throttle for *class* to *rate* bytes per second. If *rate* is 0, the throttle is disabled. If *class* is "none" or *rate* is not specified, disable this.

rateput *class* [*rate*]
Set the maximum put (**STOR**) transfer rate throttle for *class* to *rate* bytes per second. If *rate* is 0, the throttle is disabled. If *class* is "none" or *rate* is not specified, disable this.

readsize *class* [*size*]
Set the size of the read buffer to read(2) a file. The default is the file system block size. This option affects only binary transfers. If *class* is “none” or *size* is not specified, use the default.

recvbufsize *class* [*size*]
Set the size of the socket receive buffer. The default is zero and the system default value will be used. This option affects only passive transfers. If *class* is “none” or *size* is not specified, use the default.

sanenames *class* [**off**]
If *class* is “none” or **off** is specified, allow uploaded file names to contain any characters valid for a file name. Otherwise, only permit file names which don’t start with a ‘.’ and only comprise of characters from the set “[-+,._A-Za-z0-9]”.

sendbufsize *class* [*size*]
Set the size of the socket send buffer. The default is zero and the system default value will be used. This option affects only binary transfers. If *class* is “none” or *size* is not specified, use the default.

sendlowat *class* [*size*]
Set the low water mark of socket send buffer. The default is zero and system default value will be used. This option affects only for binary transfer. If *class* is “none” or *size* is not specified, use the default.

template *class* [*refclass*]
Define *refclass* as the ‘template’ for *class*; any reference to *refclass* in following directives will also apply to members of *class*. This is useful to define a template class so that other classes which are to share common attributes can be easily defined without unnecessary duplication. There can be only one template defined at a time. If *refclass* is not specified, disable the template for *class*.

timeout *class* [*time*]
Set the inactivity timeout period. (the default is fifteen minutes). This cannot be less than 30 seconds, or greater than the value for **maxtimeout**. If *class* is “none” or *time* is not specified, use the default.

umask *class* [*umaskval*]
Set the umask to *umaskval*. If *class* is “none” or *umaskval* is not specified, set to the default of 027.

upload *class* [**off**]
If *class* is “none” or **off** is specified, disable the following commands: **APPE**, **STOR**, and **STOU**, as well as the modify commands: **CHMOD**, **DELE**, **MKD**, **RMD**, **RNFR**, and **UMASK**. Otherwise, enable them.

writesize *class* [*size*]
Limit the number of bytes to write(2) at a time. The default is zero, which means all the data available as a result of mmap(2) or read(2) will be written at a time. This option affects only binary transfers. If *class* is “none” or *size* is not specified, use the default.

Numeric argument suffix parsing
Where command arguments are numeric, a decimal number is expected. Two or more numbers may be separated by an “x” to indicate a product. Each number may have one of the following optional suffixes:

b Block; multiply by 512
k Kibi; multiply by 1024 (1 KiB)

m Mebi; multiply by 1048576 (1 MiB)
g Gibi; multiply by 1073741824 (1 GiB)
t Tebi; multiply by 1099511627776 (1 TiB)
w Word; multiply by the number of bytes in an integer

See `strsuftoll`(3) for more information.

DEFAULTS

The following defaults are used:

```
checkportcmd  all
classtype     chroot CHROOT
classtype     guest  GUEST
classtype     real   REAL
display       none
limit         all    -1     # unlimited connections
maxtimeout    all    7200   # 2 hours
modify        all
motd          all    motd
notify        none
passive       all
timeout       all    900    # 15 minutes
umask         all    027
upload        all
modify        guest  off
umask         guest  0707
```

FILES

`/etc/ftpd.conf`	This file.
`/usr/share/examples/ftpd/ftpd.conf`	A sample **`ftpd.conf`** file.

SEE ALSO

`strsuftoll`(3), `ftpchroot`(5), `ftpusers`(5), `ftpd`(8)

HISTORY

The **`ftpd.conf`** functionality was implemented in NetBSD 1.3 and later releases by Luke Mewburn, based on work by Simon Burge.

NAME

`/etc/ftpusers` — FTP access list file

DESCRIPTION

`/etc/ftpusers` contains a list of users that should be allowed or denied FTP access. Each line contains a user, optionally followed by “allow” (anything but “allow” is ignored). The semi-user “∗” matches any user. Users that has an explicit “allow”, or that does not match any line, are allowed access. Anyone else is denied access.

Note that this is compatible with the old format, where this file contained a list of users that should be denied access.

EXAMPLES

This will deny anyone but “foo” and “bar” to use FTP:

```
foo allow
bar allow
*
```

SEE ALSO

`ftpd`(8)

NAME

ftpusers, **ftpchroot** — ftpd(8) access control file

DESCRIPTION

The **ftpusers** file provides user access control for ftpd(8) by defining which users may login.

If the **ftpusers** file does not exist, all users are denied access.

A "\" is the escape character; it can be used to escape the meaning of the comment character, or if it is the last character on a line, extends a configuration directive across multiple lines. A "#" is the comment character, and all characters from it to the end of line are ignored (unless it is escaped with the escape character).

The syntax of each line is:

```
userglob[:groupglob][@host] [directive [class]]
```

These elements are:

userglob matched against the user name, using fnmatch(3) glob matching (e.g, 'f*').

groupglob matched against all the groups that the user is a member of, using fnmatch(3) glob matching (e.g, '*src').

host either a CIDR address (refer to inet_net_pton(3)) to match against the remote address (e.g, '1.2.3.4/24'), or an fnmatch(3) glob to match against the remote hostname (e.g, '*.NetBSD.org').

directive If "allow" or "yes" the user is allowed access. If "deny" or "no", or **directive** is not given, the user is denied access.

class defines the class to use in ftpd.conf(5).

If **class** is not given, it defaults to one of the following:

chroot If there is a match in **/etc/ftpchroot** for the user.

guest If the user name is "anonymous" or 'ftp'.

real If neither of the above is true.

No further comparisons are attempted after the first successful match. If no match is found, the user is granted access. This syntax is backward-compatible with the old syntax.

If a user requests a guest login, the ftpd(8) server checks to see that both "anonymous" and "ftp" have access, so if you deny all users by default, you will need to add both "anonymous allow" and "ftp allow" to /etc/ftpusers in order to allow guest logins.

/etc/ftpchroot

The file /etc/ftpchroot is used to determine which users will have their session's root directory changed (using chroot(2)), either to the directory specified in the ftpd.conf(5) **chroot** directive (if set), or to the home directory of the user. If the file does not exist, the root directory change is not performed.

The syntax is similar to **ftpusers**, except that the **class** argument is ignored. If there's a positive match, the session's root directory is changed. No further comparisons are attempted after the first successful match. This syntax is backward-compatible with the old syntax.

FILES

/etc/ftpchroot	List of normal users who should have their ftp session's root directory changed by using chroot(2).

`/etc/ftpusers`	This file.
`/usr/share/examples/ftpd/ftpusers`	A sample **`ftpusers`** file.

SEE ALSO

`fnmatch`(3), `inet_net_pton`(3), `ftpd.conf`(5), `ftpd`(8)

NAME

genassym.cf — assym.h definition file

DESCRIPTION

The **genassym.cf** file is used by genassym(1) to make constant C expressions known to assembler source files. Lines starting with '#' are discarded by genassym(1). Lines starting with *include*, *ifdef*, *if*, *else* or *endif* are preceded with '#' and passed otherwise unmodified to the C compiler. Lines starting with *quote* get passed on with the *quote* command removed. The first word after a *define* command is taken as a CPP identifier and the rest of the line has to be a constant C expression. The output of genassym(1) will assign the numerical value of this expression to the CPP identifier. *export X* is a shorthand for *define X X*. *struct X* remembers X for the *member* command and does a *define X_SIZEOF sizeof(X)*. *member X* does a *define X offsetof(<last struct>, X)*. *config <ctype> <gcc constraint> <asm print modifier>* can be used to customize the output of genassym(1). When producing C output, values are casted to <ctype> (default: long) before they get handed to printf. <gcc constraint> (default: n) is the constraint used in the __asm__ statements. <asm print modifier> (default: empty) can be used to force gcc to output operands in different ways then normal. The "a" modifier e.g. stops gcc from emitting immediate prefixes in front of constants for the i386 and m68k port.

FILES

`/usr/src/sys/arch/${MACHINE}/${MACHINE}/genassym.cf`

SEE ALSO

genassym(1)

HISTORY

The **genassym.cf** file appeared in NetBSD 1.3.

NAME

generic − Postfix generic table format

SYNOPSIS

postmap /etc/postfix/generic

postmap -q "*string***" /etc/postfix/generic**

postmap -q - /etc/postfix/generic <*inputfile*

DESCRIPTION

The optional **generic**(5) table specifies an address mapping that applies when mail is delivered. This is the opposite of **canonical**(5) mapping, which applies when mail is received.

Typically, one would use the **generic**(5) table on a system that does not have a valid Internet domain name and that uses something like *localdomain.local* instead. The **generic**(5) table is then used by the **smtp**(8) client to transform local mail addresses into valid Internet mail addresses when mail has to be sent across the Internet. See the EXAMPLE section at the end of this document.

The **generic**(5) mapping affects both message header addresses (i.e. addresses that appear inside messages) and message envelope addresses (for example, the addresses that are used in SMTP protocol commands).

Normally, the **generic**(5) table is specified as a text file that serves as input to the **postmap**(1) command. The result, an indexed file in **dbm** or **db** format, is used for fast searching by the mail system. Execute the command "**postmap /etc/postfix/generic**" to rebuild an indexed file after changing the corresponding text file.

When the table is provided via other means such as NIS, LDAP or SQL, the same lookups are done as for ordinary indexed files.

Alternatively, the table can be provided as a regular-expression map where patterns are given as regular expressions, or lookups can be directed to TCP-based server. In those case, the lookups are done in a slightly different way as described below under "REGULAR EXPRESSION TABLES" or "TCP-BASED TABLES".

CASE FOLDING

The search string is folded to lowercase before database lookup. As of Postfix 2.3, the search string is not case folded with database types such as regexp: or pcre: whose lookup fields can match both upper and lower case.

TABLE FORMAT

The input format for the **postmap**(1) command is as follows:

pattern result
: When *pattern* matches a mail address, replace it by the corresponding *result*.

blank lines and comments
: Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

multi-line text
: A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

TABLE SEARCH ORDER

With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, patterns are tried in the order as listed below:

user@domain address
Replace *user@domain* by *address*. This form has the highest precedence.

user address
Replace *user@site* by *address* when *site* is equal to $**myorigin**, when *site* is listed in $**mydestination**, or when it is listed in $**inet_interfaces** or $**proxy_interfaces**.

@domain address
Replace other addresses in *domain* by *address*. This form has the lowest precedence.

RESULT ADDRESS REWRITING

The lookup result is subject to address rewriting:

- When the result has the form *@otherdomain*, the result becomes the same *user* in *otherdomain*.
- When "**append_at_myorigin=yes**", append "**@$myorigin**" to addresses without "@domain".
- When "**append_dot_mydomain=yes**", append "**.$mydomain**" to addresses without ".domain".

ADDRESS EXTENSION

When a mail address localpart contains the optional recipient delimiter (e.g., *user+foo@domain*), the lookup order becomes: *user+foo@domain*, *user@domain*, *user+foo*, *user*, and *@domain*.

The **propagate_unmatched_extensions** parameter controls whether an unmatched address extension (*+foo*) is propagated to the result of table lookup.

REGULAR EXPRESSION TABLES

This section describes how the table lookups change when the table is given in the form of regular expressions. For a description of regular expression lookup table syntax, see **regexp_table**(5) or **pcre_table**(5).

Each pattern is a regular expression that is applied to the entire address being looked up. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Patterns are applied in the order as specified in the table, until a pattern is found that matches the search string.

Results are the same as with indexed file lookups, with the additional feature that parenthesized substrings from the pattern can be interpolated as **$1**, **$2** and so on.

TCP-BASED TABLES

This section describes how the table lookups change when lookups are directed to a TCP-based server. For a description of the TCP client/server lookup protocol, see **tcp_table**(5). This feature is not available up to and including Postfix version 2.4.

Each lookup operation uses the entire address once. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Results are the same as with indexed file lookups.

EXAMPLE

The following shows a generic mapping with an indexed file. When mail is sent to a remote host via SMTP, this replaces *his@localdomain.local* by his ISP mail address, replaces *her@localdomain.local* by her ISP mail address, and replaces other local addresses by his ISP account, with an address extension of *+local* (this example assumes that the ISP supports "+" style address extensions).

/etc/postfix/main.cf:

```
smtp_generic_maps = hash:/etc/postfix/generic
```

/etc/postfix/generic:

```
his@localdomain.local   hisaccount@hisisp.example
her@localdomain.local   heraccount@herisp.example
@localdomain.local      hisaccount+local@hisisp.example
```

Execute the command "**postmap /etc/postfix/generic**" whenever the table is changed. Instead of **hash**, some systems use **dbm** database files. To find out what tables your system supports use the command "**postconf -m**".

BUGS

The table format does not understand quoting conventions.

CONFIGURATION PARAMETERS

The following **main.cf** parameters are especially relevant. The text below provides only a parameter summary. See **postconf**(5) for more details including examples.

smtp_generic_maps
: Address mapping lookup table for envelope and header sender and recipient addresses while delivering mail via SMTP.

propagate_unmatched_extensions
: A list of address rewriting or forwarding mechanisms that propagate an address extension from the original address to the result. Specify zero or more of **canonical**, **virtual**, **alias**, **forward**, **include**, or **generic**.

Other parameters of interest:

inet_interfaces
: The network interface addresses that this system receives mail on. You need to stop and start Postfix when this parameter changes.

proxy_interfaces
: Other interfaces that this machine receives mail on by way of a proxy agent or network address translator.

mydestination
: List of domains that this mail system considers local.

myorigin
: The domain that is appended to locally-posted mail.

owner_request_special
: Give special treatment to **owner-***xxx* and *xxx*-**request** addresses.

SEE ALSO

postmap(1), Postfix lookup table manager
postconf(5), configuration parameters
smtp(8), Postfix SMTP client

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
ADDRESS_REWRITING_README, address rewriting guide
DATABASE_README, Postfix lookup table overview
STANDARD_CONFIGURATION_README, configuration examples

LICENSE

The Secure Mailer license must be distributed with this software.

HISTORY

A genericstable feature appears in the Sendmail MTA.

This feature is available in Postfix 2.2 and later.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

gettytab — terminal configuration data base

SYNOPSIS

gettytab

DESCRIPTION

The **gettytab** file is a simplified version of the termcap(5) data base used to describe terminal lines. The initial terminal login process getty(8) accesses the **gettytab** file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

Where to run getty(8) processes is normally defined by ttys(5).

There is a default terminal class, *default*, that is used to set global defaults for all other classes. (That is, the *default* entry is read, then the entry for the class required is used to override particular settings.) The *default* entry is also normally read by other programs that present login prompts to the user, such as telnetd(8), in order to retrieve the values of the *he*, *hn*, *im*, and *if* capabilities.

CAPABILITIES

Refer to termcap(5) for a description of the file layout. The *default* column below lists defaults obtained if there is no entry in the table obtained, nor one in the special *default* table.

Name	Type	Default	Description
ab	bool	false	Auto-baud speed select mechanism for the Micom 600 portselector. Selection is done by looking at how the character '\r' is garbled at 300, 1200, 4800, and 9600 baud.
al	str	NULL	user to auto-login instead of prompting
ap	bool	false	terminal uses any parity
bk	str	0377	alternative end of line character (input break)
b2	str	0377	alternative end of line character (input break)
c0	num	unused	tty control flags to write messages
c1	num	unused	tty control flags to read login name
c2	num	unused	tty control flags to leave terminal as
ce	bool	false	use crt erase algorithm
ck	bool	false	use crt kill algorithm
cl	str	NULL	screen clear sequence
co	bool	false	console - add \r\n after login prompt
cs	bool	false	clear screen based on terminal type in /etc/ttys
ds	str	'^Y'	delayed suspend character
dx	bool	false	set DECCTLQ
ec	bool	false	leave echo OFF
ep	bool	false	terminal uses even parity
er	str	'^?'	erase character
et	str	'^D'	end of text (EOF) character
ev	str	NULL	initial environment
f0	num	unused	tty mode flags to write messages
f1	num	unused	tty mode flags to read login name
f2	num	unused	tty mode flags to leave terminal as
fl	str	'^O'	output flush character
hc	bool	false	do NOT hangup line on last close

he	str	NULL	hostname editing string
hn	str	hostname	hostname
ht	bool	false	terminal has real tabs
i0	num	unused	tty input flags to write messages
i1	num	unused	tty input flags to read login name
i2	num	unused	tty input flags to leave terminal as
if	str	NULL	display named file before prompt, like /etc/issue
ig	bool	false	ignore garbage characters in login name
im	str	NULL	initial (banner) message
in	str	‘^C’	interrupt character
is	num	unused	input speed
kl	str	‘^U’	kill character
l0	num	unused	tty local flags to write messages
l1	num	unused	tty local flags to read login name
l2	num	unused	tty local flags to leave terminal as
lc	bool	false	terminal has lower case
lm	str	login:	login prompt
ln	str	‘^V’	‘‘literal next’’ character
lo	str	/usr/bin/login	program to exec when name obtained
mb	bool	false	do flow control based on carrier
nl	bool	false	terminal has (or might have) a newline character
nn	bool	false	do not prompt for a login name
np	bool	false	terminal uses no parity (i.e. 8-bit characters)
nx	str	default	next table (for auto speed selection)
o0	num	unused	tty output flags to write messages
o1	num	unused	tty output flags to read login name
o2	num	unused	tty output flags to leave terminal as
op	bool	false	terminal uses odd parity
os	num	unused	output speed
pc	str	‘\0’	pad character
pe	bool	false	use printer (hard copy) erase algorithm
pf	num	0	delay between first prompt and following flush (seconds)
pp	str	unused	PPP authentication program
ps	bool	false	line connected to a MICOM port selector
qu	str	‘^\’	quit character
rp	str	‘^R’	line retype character
rw	bool	false	do NOT use raw for input, use cbreak
sp	num	unused	line speed (input and output)
st	str	‘^T’	status character
su	str	‘^Z’	suspend character
tc	str	none	table continuation
to	num	0	timeout (seconds)
tt	str	NULL	terminal type (for environment)
ub	bool	false	do unbuffered output (of prompts etc)
we	str	‘^W’	word erase character
xc	bool	false	do NOT echo control chars as ‘^X’
xf	str	‘^S’	XOFF (stop output) character
xn	str	‘^Q’	XON (start output) character

The following capabilities are no longer supported by getty(8):

bd	num	0	backspace delay
cb	bool	false	use crt backspace mode
cd	num	0	carriage-return delay
fd	num	0	form-feed (vertical motion) delay
nd	num	0	newline (line-feed) delay
uc	bool	false	terminal is known upper case only

If no line speed is specified, speed will not be altered from that which prevails when getty is entered. Specifying an input or output speed will override line speed for stated direction only.

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overridden with one of the *c0*, *c1*, *c2*, *i0*, *i1*, *i2*, *l0*, *l1*, *l2*, *o0*, *o1*, or *o2* numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. These flags correspond to the termios *c_cflag*, *c_iflag*, *c_lflag*, and *c_oflag* fields, respectively. Each these sets must be completely specified to be effective. The *f0*, *f1*, and *f2* are excepted for backwards compatibility with a previous incarnation of the TTY sub-system. In these flags the bottom 16 bits of the (32 bits) value contain the sgttyb *sg_flags* field, while the top 16 bits represent the local mode word.

Should `getty`(8) receive a null character (presumed to indicate a line break) it will restart using the table indicated by the *nx* entry. If there is none, it will re-use its original table.

Delays are specified in milliseconds, the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The *cl* screen clear string may be preceded by a (decimal) number of milliseconds of delay required (a la termcap). This delay is simulated by repeated use of the pad character *pc*.

The initial message, and login message, *im* and *lm* may include any of the following character sequences, which expand to information about the environment in which `getty`(8) is running.

%d The current date.

%h The hostname of the machine, which is normally obtained from the system using `gethostname`(3), but may also be overridden by the *hn* table entry. In either case it may be edited with the *he* string. A '@' in the *he* string causes one character from the real hostname to be copied to the final hostname. A '#' in the *he* string causes the next character of the real hostname to be skipped. Each character that is neither '@' nor '#' is copied into the final hostname. Surplus '@' and '#' characters are ignored.

%t The tty name.

%m, %r, %s, %v
The type of machine, release of the operating system, name of the operating system, and version of the kernel, respectively, as returned by `uname`(3).

%% A "%" character.

When getty execs the login process, given in the *lo* string (usually "`/usr/bin/login`"), it will have set the environment to include the terminal type, as indicated by the *tt* string (if it exists). The *ev* string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form *name=value*.

If a non-zero timeout is specified, with *to*, then getty will exit within the indicated number of seconds, either having received a login name and passed control to `login`(1), or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from `getty`(8) is even parity unless *op* or *np* is specified. The *op* string may be specified with *ap* to allow any parity on input, but generate odd parity output. Note: this only applies while getty is being run, terminal driver limitations prevent a more complete implementation. `getty`(8) does not check parity of

input characters in RAW mode.

If *pp* string is specified and a Point to Point Protocol (PPP) link bringup sequence is recognized, getty(8) will invoke the program referenced by the *pp* string, e.g. pppd(8). This can be used to handle incoming PPP calls.

SEE ALSO

login(1), gethostname(3), uname(3), termcap(5), ttys(5), getty(8), pppd(8), telnetd(8)

HISTORY

The **gettytab** file format appeared in 4.2BSD.

BUGS

The special characters (erase, kill, etc.) are reset to system defaults by login(1). In *all* cases, '#' or '^H' typed in a login name will be treated as an erase character, and '@' will be treated as a kill character.

The delay stuff is a real crock. Apart from its general lack of flexibility, some of the delay algorithms are not implemented. The terminal driver should support sane delay settings.

The *he* capability is stupid.

The termcap(5) format is horrid, something more rational should have been chosen.

NAME

group — format of the group permissions file

DESCRIPTION

The **group** file /etc/group is the local source of group information. It can be used in conjunction with the Hesiod domain 'group', and the NIS maps 'group.byname' and 'group.bygid', as controlled by nsswitch.conf(5).

The **group** file consists of newline separated ASCII records, usually one per group, containing four colon ':' separated fields. Each line has the form:

```
group:passwd:gid:[member[,member]...]
```

These fields are as follows:

group	Name of the group.
passwd	Group's *encrypted* password.
gid	The group's decimal ID.
member	Group members.

The *group* field is the group name used for granting file access to users who are members of the group.

The *gid* field is the number associated with the group name. They should both be unique across the system (and often across a group of systems) since they control file access.

The *passwd* field is an optional *encrypted* password. This field is rarely used and an asterisk is normally placed in it rather than leaving it blank.

The *member* field contains the names of users granted the privileges of *group*. The member names are separated by commas without spaces or newlines. A user is automatically in a group if that group was specified in their /etc/passwd entry and does not need to be added to that group in the /etc/group file.

Very large groups can be accommodated over multiple lines by specifying the same group name in all of them; other than this, each line has an identical format to that described above. This can be necessary to avoid the record's length limit, which is currently set to 1024 characters. Note that the limit can be queried through sysconf(3) by using the _SC_GETGR_R_SIZE_MAX parameter. For example:

```
biggrp:*:1000:user001,user002,user003,...,user099,user100
biggrp:*:1000:user101,user102,user103,...
```

The group with the name "wheel" has a special meaning to the su(1) command: if it exists and has any members, only users listed in that group are allowed to **su** to "root".

HESIOD SUPPORT

If 'dns' is specified for the 'group' database in nsswitch.conf(5), then **group** lookups occur from the 'group' Hesiod domain.

NIS SUPPORT

If 'nis' is specified for the 'group' database in nsswitch.conf(5), then **group** lookups occur from the 'group.byname' and 'group.bygid' NIS map.

COMPAT SUPPORT

If 'compat' is specified for the 'group' database, and either 'dns' or 'nis' is specified for the 'group_compat' database in nsswitch.conf(5), then the **group** file may also contain lines of the format

+name:*::

which causes the specified group to be included from the 'group' Hesiod domain or the 'group.byname' NIS map (respectively).

If no group name is specified, or the plus sign (“+”) appears alone on line, all groups are included from the Hesiod domain or the NIS map.

Hesiod or NIS compat references may appear anywhere in the file, but the single plus sign (“+”) form should be on the last line, for historical reasons. Only the first group with a specific name encountered, whether in the **group** file itself, or included via Hesiod or NIS, will be used.

FILES

`/etc/group`

SEE ALSO

`passwd`(1), `newgrp`(1), `su`(1), `setgroups`(2), `crypt`(3), `initgroups`(3), `nsswitch.conf`(5), `passwd`(5), `yp`(8)

HISTORY

A **group** file format appeared in Version 6 AT&T UNIX.

The NIS file format first appeared in SunOS.

The Hesiod support first appeared in NetBSD 1.4.

BUGS

The `passwd`(1) command does not change the **group** passwords.

NAME

header_checks − Postfix built-in content inspection

SYNOPSIS

header_checks = pcre:/etc/postfix/header_checks
mime_header_checks = pcre:/etc/postfix/mime_header_checks
nested_header_checks = pcre:/etc/postfix/nested_header_checks
body_checks = pcre:/etc/postfix/body_checks

postmap -q "*string***" pcre:/etc/postfix/***filename*
postmap -q - pcre:/etc/postfix/*filename* <*inputfile*

DESCRIPTION

This document describes access control on the content of message headers and message body lines; it is implemented by the Postfix **cleanup**(8) server before mail is queued. See **access**(5) for access control on remote SMTP client information.

Each message header or message body line is compared against a list of patterns. When a match is found the corresponding action is executed, and the matching process is repeated for the next message header or message body line.

For examples, see the EXAMPLES section at the end of this manual page.

Postfix header or body_checks are designed to stop a flood of mail from worms or viruses; they do not decode attachments, and they do not unzip archives. See the documents referenced below in the README FILES section if you need more sophisticated content analysis.

Postfix supports four built-in content inspection classes:

header_checks
: These are applied to initial message headers (except for the headers that are processed with **mime_header_checks**).

mime_header_checks (default: **$header_checks**)
: These are applied to MIME related message headers only.

  This feature is available in Postfix 2.0 and later.

nested_header_checks (default: **$header_checks**)
: These are applied to message headers of attached email messages (except for the headers that are processed with **mime_header_checks**).

  This feature is available in Postfix 2.0 and later.

body_checks
: These are applied to all other content, including multi-part message boundaries.

  With Postfix versions before 2.0, all content after the initial message headers is treated as body content.

Note: message headers are examined one logical header at a time, even when a message header spans multiple lines. Body lines are always examined one line at a time.

COMPATIBILITY

With Postfix version 2.2 and earlier specify "**postmap -fq**" to query a table that contains case sensitive patterns. By default, regexp: and pcre: patterns are case insensitive.

TABLE FORMAT

This document assumes that header and body_checks rules are specified in the form of Postfix regular expression lookup tables. Usually the best performance is obtained with **pcre** (Perl Compatible Regular

Expression) tables, but the slower **regexp** (POSIX regular expressions) support is more widely available. Use the command "**postconf -m**" to find out what lookup table types your Postfix system supports.

The general format of Postfix regular expression tables is given below. For a discussion of specific pattern or flags syntax, see **pcre_table**(5) or **regexp_table**(5), respectively.

/*pattern*/*flags action*
When *pattern* matches the input string, execute the corresponding *action*. See below for a list of possible actions.

!/*pattern*/*flags action*
When *pattern* does **not** match the input string, execute the corresponding *action*.

if /*pattern*/*flags*

endif Match the input string against the patterns between **if** and **endif**, if and only if the same input string also matches *pattern*. The **if**..**endif** can nest.

Note: do not prepend whitespace to patterns inside **if**..**endif**.

if !/*pattern*/*flags*

endif Match the input string against the patterns between **if** and **endif**, if and only if the same input string does **not** match *pattern*. The **if**..**endif** can nest.

blank lines and comments
Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a '#'.

multi-line text
A pattern/action line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

TABLE SEARCH ORDER

For each line of message input, the patterns are applied in the order as specified in the table. When a pattern is found that matches the input line, the corresponding action is executed and then the next input line is inspected.

TEXT SUBSTITUTION

Substitution of substrings from the matched expression into the *action* string is possible using the conventional Perl syntax (**$1**, **$2**, etc.). The macros in the result string may need to be written as **${n}** or **$(n)** if they aren't followed by whitespace.

Note: since negated patterns (those preceded by **!**) return a result when the expression does not match, substitutions are not available for negated patterns.

ACTIONS

Action names are case insensitive. They are shown in upper case for consistency with other Postfix documentation.

DISCARD *optional text...*
Claim successful delivery and silently discard the message. Log the optional text if specified, otherwise log a generic message.

Note: this action disables further header or body_checks inspection of the current message and affects all recipients. To discard only one recipient without discarding the entire message, use the transport(5) table to direct mail to the discard(8) service.

This feature is available in Postfix 2.0 and later.

DUNNO

Pretend that the input line did not match any pattern, and inspect the next input line. This action can be used to shorten the table search.

For backwards compatibility reasons, Postfix also accepts **OK** but it is (and always has been) treated as **DUNNO**.

This feature is available in Postfix 2.1 and later.

FILTER *transport:destination*

Write a content filter request to the queue file, and inspect the next input line. After the complete message is received it will be sent through the specified external content filter. More information about external content filters is in the Postfix FILTER_README file.

Note: this action overrides the **content_filter** setting, and affects all recipients of the message. In the case that multiple **FILTER** actions fire, only the last one is executed.

This feature is available in Postfix 2.0 and later.

HOLD *optional text...*

Arrange for the message to be placed on the **hold** queue, and inspect the next input line. The message remains on **hold** until someone either deletes it or releases it for delivery. Log the optional text if specified, otherwise log a generic message.

Mail that is placed on hold can be examined with the **postcat**(1) command, and can be destroyed or released with the **postsuper**(1) command.

Note: use "**postsuper -r**" to release mail that was kept on hold for a significant fraction of **$maximal_queue_lifetime** or **$bounce_queue_lifetime**, or longer. Use "**postsuper -H**" only for mail that will not expire within a few delivery attempts.

Note: this action affects all recipients of the message.

This feature is available in Postfix 2.0 and later.

IGNORE

Delete the current line from the input, and inspect the next input line.

PREPEND *text...*

Prepend one line with the specified text, and inspect the next input line.

Notes:

- The prepended text is output on a separate line, immediately before the input that triggered the **PREPEND** action.
- The prepended text is not considered part of the input stream: it is not subject to header/body checks or address rewriting, and it does not affect the way that Postfix adds missing message headers.
- When prepending text before a message header line, the prepended text must begin with a valid message header label.
- This action cannot be used to prepend multi-line text.

This feature is available in Postfix 2.1 and later.

REDIRECT *user@domain*

Write a message redirection request to the queue file, and inspect the next input line. After the message is queued, it will be sent to the specified address instead of the intended recipient(s).

Note: this action overrides the **FILTER** action, and affects all recipients of the message. If multiple **REDIRECT** actions fire, only the last one is executed.

This feature is available in Postfix 2.1 and later.

REPLACE *text...*
Replace the current line with the specified text, and inspect the next input line.

This feature is available in Postfix 2.2 and later. The description below applies to Postfix 2.2.2 and later.

Notes:

- When replacing a message header line, the replacement text must begin with a valid header label.
- The replaced text remains part of the input stream. Unlike the result from the **PREPEND** action, a replaced message header may be subject to address rewriting and may affect the way that Postfix adds missing message headers.

REJECT *optional text...*
Reject the entire message. Reply with *optional text...* when the optional text is specified, otherwise reply with a generic error message.

Note: this action disables further header or body_checks inspection of the current message and affects all recipients.

Postfix version 2.3 and later support enhanced status codes. When no code is specified at the beginning of *optional text...*, Postfix inserts a default enhanced status code of "5.7.1".

WARN *optional text...*
Log a warning with the *optional text...* (or log a generic message), and inspect the next input line. This action is useful for debugging and for testing a pattern before applying more drastic actions.

BUGS

Many people overlook the main limitations of header and body_checks rules.

- These rules operate on one logical message header or one body line at a time. A decision made for one line is not carried over to the next line.
- If text in the message body is encoded (RFC 2045) then the rules need to be specified for the encoded form.
- Likewise, when message headers are encoded (RFC 2047) then the rules need to be specified for the encoded form.

Message headers added by the **cleanup**(8) daemon itself are excluded from inspection. Examples of such message headers are **From:**, **To:**, **Message-ID:**, **Date:**.

Message headers deleted by the **cleanup**(8) daemon will be examined before they are deleted. Examples are: **Bcc:, Content-Length:**, **Return-Path:**.

CONFIGURATION PARAMETERS

body_checks
Lookup tables with content filter rules for message body lines. These filters see one physical line at a time, in chunks of at most **$line_length_limit** bytes.

body_checks_size_limit
The amount of content per message body segment (attachment) that is subjected to **$body_checks** filtering.

header_checks

mime_header_checks (default: **$header_checks**)

nested_header_checks (default: **$header_checks**)
Lookup tables with content filter rules for message header lines: respectively, these are applied to the initial message headers (not including MIME headers), to the MIME headers anywhere in the message, and to the initial headers of attached messages.

Note: these filters see one logical message header at a time, even when a message header spans multiple lines. Message headers that are longer than **$header_size_limit** characters are truncated.

disable_mime_input_processing
While receiving mail, give no special treatment to MIME related message headers; all text after the initial message headers is considered to be part of the message body. This means that **header_checks** is applied to all the initial message headers, and that **body_checks** is applied to the remainder of the message.

Note: when used in this manner, **body_checks** will process a multi-line message header one line at a time.

EXAMPLES

Header pattern to block attachments with bad file name extensions.

/etc/postfix/main.cf:
```
header_checks = regexp:/etc/postfix/header_checks
```

/etc/postfix/header_checks:
```
/^content-(type|disposition):.*name[[:space:]]*=.*\.(exe|vbs)/
    REJECT Bad attachment file name extension: $2
```

Body pattern to stop a specific HTML browser vulnerability exploit.

/etc/postfix/main.cf:
```
body_checks = regexp:/etc/postfix/body_checks
```

/etc/postfix/body_checks:
```
/^<iframe src=(3D)?cid:.* height=(3D)?0 width=(3D)?0>$/
    REJECT IFRAME vulnerability exploit
```

SEE ALSO

cleanup(8), canonicalize and enqueue Postfix message
pcre_table(5), format of PCRE lookup tables
regexp_table(5), format of POSIX regular expression tables
postconf(1), Postfix configuration utility
postmap(1), Postfix lookup table management
postsuper(1), Postfix janitor
postcat(1), show Postfix queue file contents
RFC 2045, base64 and quoted-printable encoding rules
RFC 2047, message header encoding for non-ASCII text

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview
CONTENT_INSPECTION_README, Postfix content inspection overview
BUILTIN_FILTER_README, Postfix built-in content inspection
BACKSCATTER_README, blocking returned forged mail

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

hesiod.conf — configuration file for the Hesiod library

DESCRIPTION

The file **hesiod.conf** determines the behavior of the Hesiod library. Blank lines and lines beginning with a '#' character are ignored. All other lines should be of the form *variable* = *value*, where the value should be a single word. Possible variables and values are:

lhs Specifies the domain prefix used for Hesiod queries. In almost all cases, you should specify "*lhs*=*.ns*". The default value if you do not specify an lhs value is no domain prefix, which is not compatible with most Hesiod domains.

rhs Specifies the default Hesiod domain; this value may be overridden by the HES_DOMAIN environment variable. You must specify an *rhs* line for the Hesiod library to work properly.

classes Specifies which DNS classes Hesiod should do lookups in. Possible values are *IN* (the preferred class) and *HS* (the deprecated class, still used by some sites). You may specify both classes separated by a comma to try one class first and then the other if no entry is available in the first class. The default value of the classes variable is "*IN,HS*".

SEE ALSO

hesiod(3)

BUGS

There default value for "lhs" should probably be more reasonable.

NAME

hostapd.conf — configuration file for hostapd(8) utility

DESCRIPTION

The **hostapd.conf** utility is an authenticator for IEEE 802.11 networks. It provides full support for WPA/IEEE 802.11i and can also act as an IEEE 802.1X Authenticator with a suitable backend Authentication Server (typically FreeRADIUS).

The configuration file consists of global parameters and domain specific configuration:

- IEEE 802.1X-2004
- RADIUS client
- RADIUS authentication server
- WPA/IEEE 802.11i

GLOBAL PARAMETERS

The following parameters are recognized:

interface Interface name. Should be set in "hostap" mode.

debug Debugging mode: 0 = no, 1 = minimal, 2 = verbose, 3 = msg dumps, 4 = excessive.

dump_file
Dump file for state information (on SIGUSR1).

ctrl_interface
The pathname of the directory in which hostapd(8) creates UNIX domain socket files for communication with frontend programs such as hostapd_cli(8).

ctrl_interface_group
A group name or group ID to use in setting protection on the control interface file. This can be set to allow non-root users to access the control interface files. If no group is specified, the group ID of the control interface is not modified and will, typically, be the group ID of the directory in which the socket is created.

IEEE 802.1X-2004 PARAMETERS

The following parameters are recognized:

ieee8021x
Require IEEE 802.1X authorization.

eap_message
Optional displayable message sent with EAP Request-Identity.

wep_key_len_broadcast
Key lengths for broadcast keys.

wep_key_len_unicast
Key lengths for unicast keys.

wep_rekey_period
Rekeying period in seconds.

eapol_key_index_workaround
EAPOL-Key index workaround (set bit7) for WinXP Supplicant.

eap_reauth_period
EAP reauthentication period in seconds. To disable reauthentication, use "0".

RADIUS CLIENT PARAMETERS

The following parameters are recognized:

own_ip_addr
The own IP address of the access point (used as NAS-IP-Address).

nas_identifier
Optional NAS-Identifier string for RADIUS messages.

auth_server_addr, *auth_server_port*, *auth_server_shared_secret*
RADIUS authentication server parameters. Can be defined twice for secondary servers to be used if primary one does not reply to RADIUS packets.

acct_server_addr, *acct_server_port*, *acct_server_shared_secret*
RADIUS accounting server parameters. Can be defined twice for secondary servers to be used if primary one does not reply to RADIUS packets.

radius_retry_primary_interval
Retry interval for trying to return to the primary RADIUS server (in seconds).

radius_acct_interim_interval
Interim accounting update interval. If this is set (larger than 0) and acct_server is configured, `hostapd`(8) will send interim accounting updates every N seconds.

RADIUS AUTHENTICATION SERVER PARAMETERS

The following parameters are recognized:

radius_server_clients
File name of the RADIUS clients configuration for the RADIUS server. If this is commented out, RADIUS server is disabled.

radius_server_auth_port
The UDP port number for the RADIUS authentication server.

radius_server_ipv6
Use IPv6 with RADIUS server.

WPA/IEEE 802.11i PARAMETERS

The following parameters are recognized:

wpa Enable WPA. Setting this variable configures the AP to require WPA (either WPA-PSK or WPA-RADIUS/EAP based on other configuration).

wpa_psk, *wpa_passphrase*
WPA pre-shared keys for WPA-PSK. This can be either entered as a 256-bit secret in hex format (64 hex digits), wpa_psk, or as an ASCII passphrase (8..63 characters) that will be converted to PSK. This conversion uses SSID so the PSK changes when ASCII passphrase is used and the SSID is changed.

wpa_psk_file
Optionally, WPA PSKs can be read from a separate text file (containing a list of (PSK,MAC address) pairs.

wpa_key_mgmt
Set of accepted key management algorithms (WPA-PSK, WPA-EAP, or both).

wpa_pairwise
Set of accepted cipher suites (encryption algorithms) for pairwise keys (unicast packets). See the example file for more information.

wpa_group_rekey
Time interval for rekeying GTK (broadcast/multicast encryption keys) in seconds.

wpa_strict_rekey
Rekey GTK when any STA that possesses the current GTK is leaving the BSS.

wpa_gmk_rekey
Time interval for rekeying GMK (master key used internally to generate GTKs (in seconds).

SEE ALSO

`hostapd`(8), `hostapd_cli`(8), `/usr/share/examples/hostapd/hostapd.conf`

HISTORY

The **`hostapd.conf`** manual page and `hostapd`(8) functionality first appeared in NetBSD 4.0.

AUTHORS

This manual page is derived from the `README` and `hostapd.conf` files in the **`hostapd`** distribution provided by Jouni Malinen ⟨jkmaline@cc.hut.fi⟩.

NAME

hosts — host name data base

DESCRIPTION

The **hosts** file contains information regarding the known hosts on the network. It can be used in conjunction with the DNS, and the NIS maps 'hosts.byaddr', and 'hosts.byname', as controlled by nsswitch.conf(5).

For each host a single line should be present with the following information:

```
address hostname [alias ...]
```

These are:

address	Internet address
hostname	Official host name
alias	Alias host name

Items are separated by any number of blanks and/or tab characters. A hash sign ("#") indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

When using the name server named(8), or ypserv(8), this file provides a backup when the name server is not running. For the name server, it is suggested that only a few addresses be included in this file. These include address for the local interfaces that ifconfig(8) needs at boot time and a few machines on the local network.

This file may be created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts. As the data base maintained at NIC is incomplete, use of the name server is recommended for sites on the DARPA Internet.

As network addresses, both IPv4 and IPv6 addresses are allowed. IPv4 addresses are specified in the conventional dot (".") notation using the inet_pton(3) routine from the Internet address manipulation library, inet(3). IPv6 addresses are specified in the standard hex-and-colon notation. Host names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/hosts The **hosts** file resides in /etc.

SEE ALSO

gethostbyname(3), nsswitch.conf(5), ifconfig(8), named(8)

Name Server Operations Guide for BIND.

HISTORY

The **hosts** file format appeared in 4.2BSD.

NAME

hosts.equiv, **.rhosts** — trusted remote hosts and host-user pairs

DESCRIPTION

The **hosts.equiv** and **.rhosts** files list hosts and users which are "trusted" by the local host when a connection is made via `rlogind`(8), `rshd`(8), or any other server that uses `ruserok`(3). This mechanism bypasses password checks, and is required for access via `rsh`(1).

Each line of these files has the format:

hostname [username]

The *hostname* may be specified as a host name (typically a fully qualified host name in a DNS environment) or address, "`+@netgroup`" (from which only the host names are checked), or a "`+`" wildcard (allow all hosts).

The *username*, if specified, may be given as a user name on the remote host, "`+@netgroup`" (from which only the user names are checked), or a "`+`" wildcard (allow all remote users).

If a *username* is specified, only that user from the specified host may login to the local machine. If a *username* is not specified, any user may login with the same user name.

EXAMPLES

`somehost`

A common usage: users on *somehost* may login to the local host as the same user name.

`somehost username`

The user *username* on *somehost* may login to the local host. If specified in `/etc/hosts.equiv`, the user may login with only the same user name.

`+@anetgroup username`

The user *username* may login to the local host from any machine listed in the netgroup *anetgroup*.

`+`

`+ +`

Two severe security hazards. In the first case, allows a user on any machine to login to the local host as the same user name. In the second case, allows any user on any machine to login to the local host (as any user, if in `/etc/hosts.equiv`).

WARNINGS

The username checks provided by this mechanism are *not* secure, as the remote user name is received by the server unchecked for validity. Therefore this mechanism should only be used in an environment where all hosts are completely trusted.

A numeric host address instead of a host name can help security considerations somewhat; the address is then used directly by `iruserok`(3).

When a username (or netgroup, or +) is specified in `/etc/hosts.equiv`, that user (or group of users, or all users, respectively) may login to the local host as *any local user*. Usernames in `/etc/hosts.equiv` should therefore be used with extreme caution, or not at all.

A `.rhosts` file must be owned by the user whose home directory it resides in, and must be writable only by that user.

Logins as root only check root's `.rhosts` file; the `/etc/hosts.equiv` file is not checked for security. Access permitted through root's `.rhosts` file is typically only for `rsh`(1), as root must still login on the console for an interactive login such as `rlogin`(1).

FILES

`/etc/hosts.equiv`	Global trusted host-user pairs list
`~/.rhosts`	Per-user trusted host-user pairs list

SEE ALSO

`rcp`(1), `rlogin`(1), `rsh`(1), `rcmd`(3), `ruserok`(3), `netgroup`(5)

HISTORY

The **`.rhosts`** file format appeared in 4.2BSD.

BUGS

The `ruserok`(3) implementation currently skips negative entries (preceded with a "`-`" sign) and does not treat them as ''short-circuit'' negative entries.

NAME

hosts_access, hosts.allow, hosts.deny − format of host access control files

DESCRIPTION

This manual page describes a simple access control language that is based on client (host name/address, user name), and server (process name, host name/address) patterns. Examples are given at the end. The impatient reader is encouraged to skip to the EXAMPLES section for a quick introduction.

Note that in a 'stock' installation of the tcp_wrappers package, a program called *tcpd* is called from */etc/inetd.conf*, and this program performs the wrapper checks and then executes the daemon. In NetBSD *inetd*(8) has been modified to perform this check internally, and so tcpd is neither used nor supplied.

Also note that libwrap under NetBSD uses the extensions to the access control language as described in the *hosts_options*(5).

In the following text, *daemon* is the process name of a network daemon process, and *client* is the name and/or address of a host requesting service. Network daemon process names are specified in the inetd configuration file.

ACCESS CONTROL FILES

The access control software consults two files. The search stops at the first match:

- Access will be granted when a (daemon,client) pair matches an entry in the */etc/hosts.allow* file.
- Otherwise, access will be denied when a (daemon,client) pair matches an entry in the */etc/hosts.deny* file.
- Otherwise, access will be granted.

A non-existing access control file is treated as if it were an empty file. Thus, access control can be turned off by providing no access control files.

ACCESS CONTROL RULES

Each access control file consists of zero or more lines of text. These lines are processed in order of appearance. The search terminates when a match is found.

- A newline character is ignored when it is preceded by a backslash character. This permits you to break up long lines so that they are easier to edit. **WARNING:** The total length of an entry can be no more than 2047 characters long including the final newline.
- Blank lines or lines that begin with a '#´ character are ignored. This permits you to insert comments and whitespace so that the tables are easier to read.
- All other lines should satisfy the following format, things between [] being optional:

    daemon_list : client_list : option : option ...

daemon_list is a list of one or more daemon process names (argv[0] values) or wildcards (see below).

client_list is a list of one or more host names, host addresses, patterns or wildcards (see below) that will be matched against the client host name or address. When a client_list item needs to include colon character (for IPv6 addresses), the item needs to be wrapped with square bracket.

The more complex forms *daemon@host* and *user@host* are explained in the sections on server endpoint patterns and on client username lookups, respectively.

List elements should be separated by blanks and/or commas.

With the exception of NIS (YP) netgroup lookups, all access control checks are case insensitive.

PATTERNS

The access control language implements the following patterns:

- A string that begins with a '.´ character. A host name is matched if the last components of its name match the specified pattern. For example, the pattern '.tue.nl´ matches the host name 'wzv.win.tue.nl´.

- A string that ends with a '.' character. A host address is matched if its first numeric fields match the given string. For example, the pattern '131.155.' matches the address of (almost) every host on the Eindhoven University network (131.155.x.x).
- A string that begins with an '@' character is treated as an NIS (formerly YP) netgroup name. A host name is matched if it is a host member of the specified netgroup. Netgroup matches are not supported for daemon process names or for client user names.
- An expression of the form 'n.n.n.n/m.m.m.m' is interpreted as a 'net/mask' pair. A host address is matched if 'net' is equal to the bitwise AND of the address and the 'mask'. For example, the net/mask pattern '131.155.72.0/255.255.254.0' matches every address in the range '131.155.72.0' through '131.155.73.255'. Note that 'm.m.m.m' portion must always be specified.
- An expression of the form 'ipv6-addr/ipv6-mask' is interpreted as masked IPv6 address match, just like masked IPv4 address match (see above). Note that 'ipv6-mask' portion must always be specified.
- An expression of the form 'ipv6-addr/prefixlen' is interpreted as masked IPv6 address match (with mask specified by numeric prefixlen), just like masked IPv4 address match (see above). Note that 'prefixlen' portion must always be specified.

WILDCARDS

The access control language supports explicit wildcards:

ALL The universal wildcard, always matches.

LOCAL
: Matches any host whose name does not contain a dot character.

UNKNOWN
: Matches any user whose name is unknown, and matches any host whose name *or* address are unknown. This pattern should be used with care: host names may be unavailable due to temporary name server problems. A network address will be unavailable when the software cannot figure out what type of network it is talking to.

KNOWN
: Matches any user whose name is known, and matches any host whose name *and* address are known. This pattern should be used with care: host names may be unavailable due to temporary name server problems. A network address will be unavailable when the software cannot figure out what type of network it is talking to.

PARANOID
: Matches any host whose name does not match its address. Note that unlike the default mode of *tcpd*, NetBSD *inetd* does not automatically drop these requests; you must explicitly drop them in your */etc/hosts.allow* or */etc/hosts.deny* file.

{RBL}.*domain*
: Matches any host whose reversed address appears in the DNS under *domain*. The primary such domain used for blocking unsolicited commercial e-mail (spam) is '.rbl.maps.vix.com'.

OPERATORS

EXCEPT
: Intended use is of the form: 'list_1 EXCEPT list_2'; this construct matches anything that matches *list_1* unless it matches *list_2*. The EXCEPT operator can be used in daemon_lists and in client_lists. The EXCEPT operator can be nested: if the control language would permit the use of parentheses, 'a EXCEPT b EXCEPT c' would parse as '(a EXCEPT (b EXCEPT c))'.

% EXPANSIONS

The following expansions are available within some options:

%a (%A)
: The client (server) host address.

%c
: Client information: user@host, user@address, a host name, or just an address, depending on how much information is available.

%d
: The daemon process name (argv[0] value).

%h (%H)
: The client (server) host name or address, if the host name is unavailable.

%n (%N)
: The client (server) host name (or "unknown" or "paranoid").

%p
: The daemon process id.

%s
: Server information: daemon@host, daemon@address, or just a daemon name, depending on how much information is available.

%u
: The client user name (or "unknown").

%%
: Expands to a single ‘%´ character.

Characters in % expansions that may confuse the shell are replaced by underscores.

SERVER ENDPOINT PATTERNS

In order to distinguish clients by the network address that they connect to, use patterns of the form:

```
process_name@host_pattern : client_list ...
```

Patterns like these can be used when the machine has different internet addresses with different internet hostnames. Service providers can use this facility to offer FTP, GOPHER or WWW archives with internet names that may even belong to different organizations. See also the ‘twist’ option in the hosts_options(5) document. Some systems (Solaris, FreeBSD, NetBSD) can have more than one internet address on one physical interface; with other systems you may have to resort to SLIP or PPP pseudo interfaces that live in a dedicated network address space.

The host_pattern obeys the same syntax rules as host names and addresses in client_list context. Usually, server endpoint information is available only with connection-oriented services.

CLIENT USERNAME LOOKUP

When the client host supports the RFC 931 protocol or one of its descendants (TAP, IDENT, RFC 1413) the wrapper programs can retrieve additional information about the owner of a connection. Client username information, when available, is logged together with the client host name, and can be used to match patterns like:

```
daemon_list : ... user_pattern@host_pattern ...
```

The daemon wrappers can be configured at compile time to perform rule-driven username lookups (default) or to always interrogate the client host. In the case of rule-driven username lookups, the above rule would cause username lookup only when both the *daemon_list* and the *host_pattern* match.

A user pattern has the same syntax as a daemon process pattern, so the same wildcards apply (netgroup membership is not supported). One should not get carried away with username lookups, though.

- The client username information cannot be trusted when it is needed most, i.e. when the client system has been compromised. In general, ALL and (UN)KNOWN are the only user name patterns that make sense.
- Username lookups are possible only with TCP-based services, and only when the client host runs a suitable daemon; in all other cases the result is "unknown".

- A well-known UNIX kernel bug may cause loss of service when username lookups are blocked by a firewall. The wrapper README document describes a procedure to find out if your kernel has this bug.
- Username lookups may cause noticeable delays for non-UNIX users. The default timeout for username lookups is 10 seconds: too short to cope with slow networks, but long enough to irritate PC users.

Selective username lookups can alleviate the last problem. For example, a rule like:

```
daemon_list : @pcnetgroup ALL@ALL
```

would match members of the pc netgroup without doing username lookups, but would perform username lookups with all other systems.

DETECTING ADDRESS SPOOFING ATTACKS

A flaw in the sequence number generator of many TCP/IP implementations allows intruders to easily impersonate trusted hosts and to break in via, for example, the remote shell service. The IDENT (RFC 931 etc.) service can be used to detect such and other host address spoofing attacks.

Before accepting a client request, the wrappers can use the IDENT service to find out that the client did not send the request at all. When the client host provides IDENT service, a negative IDENT lookup result (the client matches 'UNKNOWN@host') is strong evidence of a host spoofing attack.

A positive IDENT lookup result (the client matches 'KNOWN@host') is less trustworthy. It is possible for an intruder to spoof both the client connection and the IDENT lookup, although doing so is much harder than spoofing just a client connection. It may also be that the client´s IDENT server is lying.

Note: IDENT lookups don´t work with UDP services.

EXAMPLES

The language is flexible enough that different types of access control policy can be expressed with a minimum of fuss. Although the language uses two access control tables, the most common policies can be implemented with one of the tables being trivial or even empty.

When reading the examples below it is important to realize that the allow table is scanned before the deny table, that the search terminates when a match is found, and that access is granted when no match is found at all.

The examples use host and domain names. They can be improved by including address and/or network/netmask information, to reduce the impact of temporary name server lookup failures.

MOSTLY CLOSED

In this case, access is denied by default. Only explicitly authorized hosts are permitted access.

The default policy (no access) is implemented with a trivial deny file:

/etc/hosts.deny:

```
ALL: ALL
```

This denies all service to all hosts, unless they are permitted access by entries in the allow file.

The explicitly authorized hosts are listed in the allow file. For example:

/etc/hosts.allow:

```
ALL: LOCAL @some_netgroup
ALL: .foobar.edu EXCEPT terminalserver.foobar.edu
```

The first rule permits access from hosts in the local domain (no '.´ in the host name) and from members of the *some_netgroup* netgroup. The second rule permits access from all hosts in the *foobar.edu* domain (notice the leading dot), with the exception of *terminalserver.foobar.edu*.

MOSTLY OPEN

Here, access is granted by default; only explicitly specified hosts are refused service.

The default policy (access granted) makes the allow file redundant so that it can be omitted. The explicitly

non-authorized hosts are listed in the deny file. For example:

/etc/hosts.deny:

```
ALL: some.host.name, .some.domain
ALL EXCEPT in.fingerd: other.host.name, .other.domain
```

The first rule denies some hosts and domains all services; the second rule still permits finger requests from other hosts and domains.

BOOBY TRAPS

The next example permits tftp requests from hosts in the local domain (notice the leading dot). Requests from any other hosts are denied. Instead of the requested file, a finger probe is sent to the offending host. The result is mailed to the superuser.

/etc/hosts.allow:

```
in.tftpd: LOCAL, .my.domain
```

/etc/hosts.deny:

```
in.tftpd: ALL: spawn (/some/where/safe_finger -l @%h | \
        /usr/ucb/mail -s %d-%h root) &
```

(The safe_finger command can be gotten from the tcp_wrappers package and installed in a suitable place. It limits possible damage from data sent by the remote finger server. It gives better protection than the standard finger command.)

The expansion of the %h (client host) and %d (service name) sequences is described in the section on shell commands.

Warning: do not booby-trap your finger daemon, unless you are prepared for infinite finger loops.

On network firewall systems this trick can be carried even further. The typical network firewall only provides a limited set of services to the outer world. All other services can be "bugged" just like the above tftp example. The result is an excellent early-warning system.

DIAGNOSTICS

An error is reported when a syntax error is found in a host access control rule; when the length of an access control rule exceeds the capacity of an internal buffer; when an access control rule is not terminated by a newline character; when the result of %<letter> expansion would overflow an internal buffer; when a system call fails that shouldn´t. All problems are reported via the syslog daemon.

FILES

/etc/hosts.allow, (daemon,client) pairs that are granted access.
/etc/hosts.deny, (daemon,client) pairs that are denied access.

SEE ALSO

hosts_options(5), hosts_access(3)
tcpdchk(8), tcpdmatch(8), test programs.

BUGS

If a name server lookup times out, the host name will not be available to the access control software, even though the host is registered.

Domain name server lookups are case insensitive; NIS (formerly YP) netgroup lookups are case sensitive.

The total length of an entry can be no more than 2047 characters long, including the final newline.

AUTHOR

Wietse Venema (wietse@wzv.win.tue.nl)
Department of Mathematics and Computing Science
Eindhoven University of Technology
Den Dolech 2, P.O. Box 513,
5600 MB Eindhoven, The Netherlands

NAME

hosts_options − host access control language extensions

DESCRIPTION

This document describes optional extensions to the language described in the hosts_access(5) document. The extensions are enabled at program build time. For example, by editing the Makefile and turning on the PROCESS_OPTIONS compile-time option.

The extensible language uses the following format:

```
daemon_list : client_list : option : option ...
```

The first two fields are described in the hosts_access(5) manual page. The remainder of the rules is a list of zero or more options. Any ":" characters within options should be protected with a backslash.

An option is of the form "keyword" or "keyword value". Options are processed in the specified order. Some options are subjected to %<letter> substitutions. For the sake of backwards compatibility with earlier versions, an "=" is permitted between keyword and value.

LOGGING

severity mail.info

severity notice

Change the severity level at which the event will be logged. Facility names (such as mail) are optional, and are not supported on systems with older syslog implementations. The severity option can be used to emphasize or to ignore specific events.

ACCESS CONTROL

allow

deny Grant (deny) service. These options must appear at the end of a rule.

The *allow* and *deny* keywords make it possible to keep all access control rules within a single file, for example in the *hosts.allow* file.

To permit access from specific hosts only:

```
ALL: .friendly.domain: ALLOW
ALL: ALL: DENY
```

To permit access from all hosts except a few trouble makers:

```
ALL: .bad.domain: DENY
ALL: ALL: ALLOW
```

Notice the leading dot on the domain name patterns.

RUNNING OTHER COMMANDS

spawn shell_command

Execute, in a child process, the specified shell command, after performing the %<letter> expansions described in the hosts_access(5) manual page. The command is executed with stdin, stdout and stderr connected to the null device, so that it won´t mess up the conversation with the client host. Example:

```
spawn (/some/where/safe_finger -l @%h | /usr/ucb/mail root) &
```

executes, in a background child process, the shell command "safe_finger -l @%h | mail root" after replacing %h by the name or address of the remote host.

The example uses the "safe_finger" command instead of the regular "finger" command, to limit

possible damage from data sent by the finger server. The "safe_finger" command is part of the daemon wrapper package; it is a wrapper around the regular finger command that filters the data sent by the remote host.

twist shell_command

Replace the current process by an instance of the specified shell command, after performing the %<letter> expansions described in the hosts_access(5) manual page. Stdin, stdout and stderr are connected to the client process. This option must appear at the end of a rule.

To send a customized bounce message to the client instead of running the real ftp daemon:

```
in.ftpd : ... : twist /bin/echo 421 Some bounce message
```

For an alternative way to talk to client processes, see the *banners* option below.

To run /some/other/in.telnetd without polluting its command-line array or its process environment:

```
in.telnetd : ... : twist PATH=/some/other; exec in.telnetd
```

Warning: in case of UDP services, do not twist to commands that use the standard I/O or the read(2)/write(2) routines to communicate with the client process; UDP requires other I/O primitives.

NETWORK OPTIONS

keepalive

Causes the server to periodically send a message to the client. The connection is considered broken when the client does not respond. The keepalive option can be useful when users turn off their machine while it is still connected to a server. The keepalive option is not useful for datagram (UDP) services.

linger number_of_seconds

Specifies how long the kernel will try to deliver not-yet delivered data after the server process closes a connection.

USERNAME LOOKUP

rfc931 [timeout_in_seconds]

Look up the client user name with the RFC 931 (TAP, IDENT, RFC 1413) protocol. This option is silently ignored in case of services based on transports other than TCP. It requires that the client system runs an RFC 931 (IDENT, etc.) -compliant daemon, and may cause noticeable delays with connections from non-UNIX clients. The timeout period is optional. If no timeout is specified a compile-time defined default value is taken.

MISCELLANEOUS

banners /some/directory

Look for a file in '/some/directory' with the same name as the daemon process (for example in.telnetd for the telnet service), and copy its contents to the client. Newline characters are replaced by carriage-return newline, and %<letter> sequences are expanded (see the hosts_access(5) manual page).

The tcp wrappers source code distribution provides a sample makefile (Banners.Makefile) for convenient banner maintenance.

Warning: banners are supported for connection-oriented (TCP) network services only.

nice [number]

Change the nice value of the process (default 10). Specify a positive value to spend more CPU resources on other processes.

setenv name value

Place a (name, value) pair into the process environment. The value is subjected to %<letter> expansions and may contain whitespace (but leading and trailing blanks are stripped off).

Warning: many network daemons reset their environment before spawning a login or shell process.

umask 022

Like the umask command that is built into the shell. An umask of 022 prevents the creation of files with group and world write permission. The umask argument should be an octal number.

user nobody

user nobody.kmem

Assume the privileges of the "nobody" userid (or user "nobody", group "kmem"). The first form is useful with inetd implementations that run all services with root privilege. The second form is useful for services that need special group privileges only.

DIAGNOSTICS

When a syntax error is found in an access control rule, the error is reported to the syslog daemon; further options will be ignored, and service is denied.

SEE ALSO

hosts_access(3) hosts_access(5), the default access control language

AUTHOR

Wietse Venema (wietse@wzv.win.tue.nl)
Department of Mathematics and Computing Science
Eindhoven University of Technology
Den Dolech 2, P.O. Box 513,
5600 MB Eindhoven, The Netherlands

NAME

ifaliases — interface aliases file

DESCRIPTION

The **ifaliases** file specifies the additional addresses (aliases) that each interface has. **ifaliases** is processed by `/etc/rc.d/network` at system boot time.

Each line of the file is of the form:

address interface [netmask]

The *address* is a network address of the alias. This must be a number, or must be in `/etc/hosts`, since the nameserver is not running at this point.

The *interface* is the network interface the alias will be configured on.

The *netmask* is the netmask of the alias' network address. Although this is optional, omitting the netmask is discouraged. Omission results in the use of the classful netmask associated with *address*.

FILES

`/etc/ifaliases`, `/etc/rc.d/network`

HISTORY

The **ifaliases** file appeared in NetBSD 1.2.

BUGS

ifaliases assumes IPv4, and does not support other protocol families. Please check `rc.conf`(5) for alternatives like `/etc/ifconfig.xxN`.

NAME

ifconfig.if — interface-specific configuration files

DESCRIPTION

The **ifconfig.if** files contain information regarding the configuration of each network interface. **ifconfig.if** is processed by /etc/rc.d/network at system boot time.

One file should exist for each interface that is to be configured, such as /etc/ifconfig.fxp0. The file will get evaluated only if the interface exists on the system. Multiple lines can be placed in a file, and will be evaluated sequentially.

Normally, a line will be evaluated as command line arguments to ifconfig(8). "ifconfig if" will be prepended on evaluation.

If the line is equal to "dhcp", dhcpcd(8) will be started for the interface.

If a line is empty, or starts with '#', the line will be ignored as comment.

If a line starts with '!', the rest of line will get evaluated as shell script fragment. Shell variables declared in /etc/rc.d/network are accessible. The most useful variable is $int, as it will be bound to the interface being configured with the file.

For example, the following illustrates static interface configuration:

```
# IPv4, with an alias
inet 10.0.1.12 netmask 255.255.255.0 media 100baseTX
inet 10.0.1.13 netmask 255.255.255.255 alias
# let us have IPv6 address on this interface
inet6 2001:db8::1 prefixlen 64 alias
# have subnet router anycast address too
inet6 2001:db8:: prefixlen 64 alias anycast
```

The following illustrates dynamic configuration setup with dhclient(8) and rtsol(8):

```
up
# autoconfigure IPv4 address
!dhclient $int
# autoconfigure IPv6 address.  Be sure to set $ip6mode to autohost.
!rtsol $int
```

The following example is for dynamically-created pseudo interfaces like gif(4):

```
up
# configure IPv6 default route toward the interface
!route add -inet6 default ::1
!route change -inet6 default -ifp $int
```

Earlier versions of /etc/rc.d/network required an explicit 'create' command for such interfaces. This is now handled automatically.

FILES

/etc/rc.d/network

SEE ALSO

rc.conf(5), ifconfig(8)

NAME

info − readable online documentation

DESCRIPTION

The Info file format is an easily-parsable representation for online documents. It can be read by *emacs(1)* and *info(1)* among other programs.

Info files are usually created from *texinfo(5)* sources by *makeinfo(1)*, but can be created from scratch if so desired.

For a full description of the Texinfo language and associated tools, please see the Texinfo manual (written in Texinfo itself). Most likely, running this command from your shell:

info texinfo

or this key sequence from inside Emacs:

M-x info RET m texinfo RET

will get you there.

AVAILABILITY

ftp://ftp.gnu.org/pub/gnu/texinfo-<version>.tar.gz
or any GNU mirror site.

REPORTING BUGS

Please send bug reports to bug-texinfo@gnu.org, general questions and discussion to help-texinfo@gnu.org.

SEE ALSO

info(1), install-info(1), makeinfo(1), texi2dvi(1),
texindex(1).
emacs(1), tex(1).
texinfo(5).

NAME

ipf, ipf.conf, ipf6.conf − IP packet filter rule syntax

DESCRIPTION

A rule file for **ipf** may have any name or even be stdin. As **ipfstat** produces parsable rules as output when displaying the internal kernel filter lists, it is quite plausible to use its output to feed back into **ipf**. Thus, to remove all filters on input packets, the following could be done:

```
# ipfstat −i | ipf −rf −
```

GRAMMAR

The format used by **ipf** for construction of filtering rules can be described using the following grammar in BNF:

```
filter-rule = [ insert ] action in-out [ options ] [ tos ] [ ttl ]
            [ proto ] ip [ group ].

insert      = "@" decnumber .
action      = block | "pass" | log | "count" | skip | auth | call .
in-out      = "in" | "out" .
options     = [ log ] [ tag ] [ "quick" ] [ "on" interface-name [ dup ]
           [ froute ] [ replyto ] ] .
tos   = "tos" decnumber | "tos" hexnumber .
ttl   = "ttl" decnumber .
proto = "proto" protocol .
ip    = srcdst [ flags ] [ with withopt ] [ icmp ] [ keep ] .
group = [ "head" decnumber ] [ "group" decnumber ] .

block = "block" [ return-icmp[return-code] | "return-rst" ] .
log   = "log" [ "body" ] [ "first" ] [ "or-block" ] [ "level" loglevel ] .
tag     = "tag" tagid .
skip  = "skip" decnumber .
auth  = "auth" | "preauth" .
call  = "call" [ "now" ] function-name .
dup   = "dup-to" interface-name [ ":" ipaddr ] .
froute      = "fastroute" | "to" interface-name [ ":" ipaddr ] .
replyto = "reply-to" interface-name [ ":" ipaddr ] .
protocol = "tcp/udp" | "udp" | "tcp" | "icmp" | decnumber .
srcdst      = "all" | fromto .
fromto      = "from" [ "!" ] object "to" [ "!" ] object .

return-icmp = "return-icmp" | "return-icmp-as-dest" .
return-code = "(" icmp-code ")" .
object      = addr [ port-comp | port-range ] .
addr  = "any" | nummask | host-name [ "mask" ipaddr | "mask" hexnumber ] .
addr  = "any" | "<thishost>" | nummask |
        host-name [ "mask" ipaddr | "mask" hexnumber ] .
port-comp = "port" compare port-num .
port-range = "port" port-num range port-num .
flags = "flags" flag { flag } [ "/" flag { flag } ] .
with  = "with" | "and" .
icmp  = "icmp-type" icmp-type [ "code" decnumber ] .
return-code = "(" icmp-code ")" .
keep  = "keep" "state" [ "(" state-options ")" ] | "keep" "frags" .
loglevel = facility"."priority | priority .

nummask     = host-name [ "/" decnumber ] .
```

```
host-name = ipaddr | hostname | "any" .
ipaddr      = host-num "." host-num "." host-num "." host-num .
host-num = digit [ digit [ digit ] ] .
port-num = service-name | decnumber .
state-options = state-opts [ "," state-options ] .

state-opts = "age" decnumber [ "/" decnumber ] | "strict" |
             "no-icmp-err" | "limit" decnumber | "newisn" | "sync" .
withopt = [ "not" | "no" ] opttype [ withopt ] .
opttype = "ipopts" | "short" | "frag" | "opt" optname .
optname     = ipopts [ "," optname ] .
ipopts  = optlist | "sec-class" [ secname ] .
secname     = seclvl [ "," secname ] .
seclvl  = "unclass" | "confid" | "reserv-1" | "reserv-2" | "reserv-3" |
        "reserv-4" | "secret" | "topsecret" .
icmp-type = "unreach" | "echo" | "echorep" | "squench" | "redir" |
          "timex" | "paramprob" | "timest" | "timestrep" | "inforeq" |
          "inforep" | "maskreq" | "maskrep"  | decnumber .
icmp-code = decumber | "net-unr" | "host-unr" | "proto-unr" | "port-unr" |
          "needfrag" | "srcfail" | "net-unk" | "host-unk" | "isolate" |
          "net-prohib" | "host-prohib" | "net-tos" | "host-tos" |
          "filter-prohib" | "host-preced" | "cutoff-preced" .
optlist     = "nop" | "rr" | "zsu" | "mtup" | "mtur" | "encode" | "ts" |
        "tr" | "sec" | "lsrr" | "e-sec" | "cipso" | "satid" | "ssrr" |
        "addext" | "visa" | "imitd" | "eip" | "finn" .
facility = "kern" | "user" | "mail" | "daemon" | "auth" | "syslog" |
         "lpr" | "news" | "uucp" | "cron" | "ftp" | "authpriv" |
         "audit" | "logalert" | "local0" | "local1" | "local2" |
         "local3" | "local4" | "local5" | "local6" | "local7" .
priority = "emerg" | "alert" | "crit" | "err" | "warn" | "notice" |
         "info" | "debug" .

hexnumber = "0" "x" hexstring .
hexstring = hexdigit [ hexstring ] .
decnumber = digit [ decnumber ] .

compare = "=" | "!=" | "<" | ">" | "<=" | ">=" | "eq" | "ne" | "lt" |
        "gt" | "le" | "ge" .
range = "<>" | "><" .
hexdigit = digit | "a" | "b" | "c" | "d" | "e" | "f" .
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .
flag  = "F" | "S" | "R" | "P" | "A" | "U" .
```

This syntax is somewhat simplified for readability, some combinations that match this grammar are disallowed by the software because they do not make sense (such as tcp **flags** for non-TCP packets).

FILTER RULES

The "briefest" valid rules are (currently) no-ops and are of the form:

block in all
pass in all
log out all
count in all

Filter rules are checked in order, with the last matching rule determining the fate of the packet (but see the **quick** option, below).

Filters are installed by default at the end of the kernel’s filter lists, prepending the rule with **@n** will cause it

to be inserted as the n'th entry in the current list. This is especially useful when modifying and testing active filter rulesets. See ipf(8) for more information.

ACTIONS

The action indicates what to do with the packet if it matches the rest of the filter rule. Each rule MUST have an action. The following actions are recognised:

block indicates that the packet should be flagged to be dropped. In response to blocking a packet, the filter may be instructed to send a reply packet, either an ICMP packet (**return-icmp**), an ICMP packet masquerading as being from the original packet's destination (**return-icmp-as-dest**), or a TCP "reset" (**return-rst**). An ICMP packet may be generated in response to any IP packet, and its type may optionally be specified, but a TCP reset may only be used with a rule which is being applied to TCP packets. When using **return-icmp** or **return-icmp-as-dest**, it is possible to specify the actual unreachable 'type'. That is, whether it is a network unreachable, port unreachable or even administratively prohibited. This is done by enclosing the ICMP code associated with it in parenthesis directly following **return-icmp** or **return-icmp-as-dest** as follows:

block return-icmp(11) ...

Would return a Type-Of-Service (TOS) ICMP unreachable error.

pass will flag the packet to be let through the filter.

log causes the packet to be logged (as described in the LOGGING section below) and has no effect on whether the packet will be allowed through the filter.

count causes the packet to be included in the accounting statistics kept by the filter, and has no effect on whether the packet will be allowed through the filter. These statistics are viewable with ipfstat(8).

call this action is used to invoke the named function in the kernel, which must conform to a specific calling interface. Customised actions and semantics can thus be implemented to supplement those available. This feature is for use by knowledgeable hackers, and is not currently documented.

skip <n>

causes the filter to skip over the next *n* filter rules. If a rule is inserted or deleted inside the region being skipped over, then the value of *n* is adjusted appropriately.

auth this allows authentication to be performed by a user-space program running and waiting for packet information to validate. The packet is held for a period of time in an internal buffer whilst it waits for the program to return to the kernel the *real* flags for whether it should be allowed through or not. Such a program might look at the source address and request some sort of authentication from the user (such as a password) before allowing the packet through or telling the kernel to drop it if from an unrecognised source.

preauth

tells the filter that for packets of this class, it should look in the pre-authenticated list for further clarification. If no further matching rule is found, the packet will be dropped (the FR_PREAUTH is not the same as FR_PASS). If a further matching rule is found, the result from that is used in its instead. This might be used in a situation where a person *logs in* to the firewall and it sets up some temporary rules defining the access for that person.

The next word must be either **in** or **out**. Each packet moving through the kernel is either inbound (just been received on an interface, and moving towards the kernel's protocol processing) or outbound (transmitted or forwarded by the stack, and on its way to an interface). There is a requirement that each filter rule explicitly state which side of the I/O it is to be used on.

OPTIONS

The list of options is brief, and all are indeed optional. Where options are used, they must be present in the order shown here. These are the currently supported options:

log indicates that, should this be the last matching rule, the packet header will be written to the **ipl** log (as described in the LOGGING section below).

tag tagid
indicates that, if this rule causes the packet to be logged or entered in the state table, the tagid will be logged as part of the log entry. This can be used to quickly match "similar" rules in scripts that post process the log files for e.g. generation of security reports or accounting purposes. The tagid is a 32 bit unsigned integer.

quick allows "short-cut" rules in order to speed up the filter or override later rules. If a packet matches a filter rule which is marked as **quick**, this rule will be the last rule checked, allowing a "short-circuit" path to avoid processing later rules for this packet. The current status of the packet (after any effects of the current rule) will determine whether it is passed or blocked.

If this option is missing, the rule is taken to be a "fall-through" rule, meaning that the result of the match (block/pass) is saved and that processing will continue to see if there are any more matches.

on allows an interface name to be incorporated into the matching procedure. Interface names are as printed by "netstat −i". If this option is used, the rule will only match if the packet is going through that interface in the specified direction (in/out). If this option is absent, the rule is taken to be applied to a packet regardless of the interface it is present on (i.e. on all interfaces). Filter rulesets are common to all interfaces, rather than having a filter list for each interface.

This option is especially useful for simple IP-spoofing protection: packets should only be allowed to pass inbound on the interface from which the specified source address would be expected, others may be logged and/or dropped.

dup-to causes the packet to be copied, and the duplicate packet to be sent outbound on the specified interface, optionally with the destination IP address changed to that specified. This is useful for off-host logging, using a network sniffer.

to causes the packet to be moved to the outbound queue on the specified interface. This can be used to circumvent kernel routing decisions, and even to bypass the rest of the kernel processing of the packet (if applied to an inbound rule). It is thus possible to construct a firewall that behaves transparently, like a filtering hub or switch, rather than a router. The **fastroute** keyword is a synonym for this option.

MATCHING PARAMETERS

The keywords described in this section are used to describe attributes of the packet to be used when determining whether rules match or don't match. The following general-purpose attributes are provided for matching, and must be used in this order:

tos packets with different Type-Of-Service values can be filtered. Individual service levels or combinations can be filtered upon. The value for the TOS mask can either be represented as a hex number or a decimal integer value.

ttl packets may also be selected by their Time-To-Live value. The value given in the filter rule must exactly match that in the packet for a match to occur. This value can only be given as a decimal integer value.

proto allows a specific protocol to be matched against. All protocol names found in **/etc/protocols** are recognised and may be used. However, the protocol may also be given as a DECIMAL number, allowing for rules to match your own protocols, or new ones which would out-date any attempted listing.

The special protocol keyword **tcp/udp** may be used to match either a TCP or a UDP packet, and has been added as a convenience to save duplication of otherwise-identical rules.

The **from** and **to** keywords are used to match against IP addresses (and optionally port numbers). Rules must specify BOTH source and destination parameters.

IP addresses may be specified in one of two ways: as a numerical address/mask, or as a hostname **mask** netmask. The hostname may either be a valid hostname, from either the hosts file or DNS (depending on your configuration and library) or of the dotted numeric form. There is no special designation for networks but network names are recognised. Note that having your filter rules depend on DNS results can introduce

an avenue of attack, and is discouraged.

There is a special case for the hostname **any** which is taken to be 0.0.0.0/0 (see below for mask syntax) and matches all IP addresses. Only the presence of "any" has an implied mask, in all other situations, a hostname MUST be accompanied by a mask. It is possible to give "any" a hostmask, but in the context of this language, it is non-sensical.

The numerical format "x/y" indicates that a mask of y consecutive 1 bits set is generated, starting with the MSB, so a y value of 16 would give 0xffff0000. The symbolic "x **mask** y" indicates that the mask y is in dotted IP notation or a hexadecimal number of the form 0x12345678. Note that all the bits of the IP address indicated by the bitmask must match the address on the packet exactly; there isn't currently a way to invert the sense of the match, or to match ranges of IP addresses which do not express themselves easily as bitmasks (anthropomorphization; it's not just for breakfast anymore).

If a **port** match is included, for either or both of source and destination, then it is only applied to TCP and UDP packets. If there is no **proto** match parameter, packets from both protocols are compared. This is equivalent to "proto tcp/udp". When composing **port** comparisons, either the service name or an integer port number may be used. Port comparisons may be done in a number of forms, with a number of comparison operators, or port ranges may be specified. When the port appears as part of the **from** object, it matches the source port number, when it appears as part of the **to** object, it matches the destination port number. See the examples for more information.

The **all** keyword is essentially a synonym for "from any to any" with no other match parameters.

Following the source and destination matching parameters, the following additional parameters may be used:

with is used to match irregular attributes that some packets may have associated with them. To match the presence of IP options in general, use **with ipopts**. To match packets that are too short to contain a complete header, use **with short**. To match fragmented packets, use **with frag**. For more specific filtering on IP options, individual options can be listed.

Before any parameter used after the **with** keyword, the word **not** or **no** may be inserted to cause the filter rule to only match if the option(s) is not present.

Multiple consecutive **with** clauses are allowed. Alternatively, the keyword **and** may be used in place of **with**, this is provided purely to make the rules more readable ("with ... and ..."). When multiple clauses are listed, all those must match to cause a match of the rule.

flags is only effective for TCP filtering. Each of the letters possible represents one of the possible flags that can be set in the TCP header. The association is as follows:

F - FIN
S - SYN
R - RST
P - PUSH
A - ACK
U - URG

The various flag symbols may be used in combination, so that "SA" would represent a SYN-ACK combination present in a packet. There is nothing preventing the specification of combinations, such as "SFR", that would not normally be generated by law-abiding TCP implementations. However, to guard against weird aberrations, it is necessary to state which flags you are filtering against. To allow this, it is possible to set a mask indicating which TCP flags you wish to compare (i.e., those you deem significant). This is done by appending "/<flags>" to the set of TCP flags you wish to match against, e.g.:

```
... flags S
                # becomes "flags S/AUPRFS" and will match
                # packets with ONLY the SYN flag set.

... flags SA
```

```
                # becomes "flags SA/AUPRFS" and will match any
                # packet with only the SYN and ACK flags set.

        ... flags S/SA
                # will match any packet with just the SYN flag set
                # out of the SYN-ACK pair; the common "establish"
                # keyword action.  "S/SA" will NOT match a packet
                # with BOTH SYN and ACK set, but WILL match "SFP".
```

icmp-type
: is only effective when used with **proto icmp** and must NOT be used in conjunction with **flags**. There are a number of types, which can be referred to by an abbreviation recognised by this language, or the numbers with which they are associated can be used. The most important from a security point of view is the ICMP redirect.

KEEP HISTORY

The second last parameter which can be set for a filter rule is whether or not to record historical information for that packet, and what sort to keep. The following information can be kept:

state keeps information about the flow of a communication session. State can be kept for TCP, UDP, and ICMP packets.

frags keeps information on fragmented packets, to be applied to later fragments.

allowing packets which match these to flow straight through, rather than going through the access control list.

GROUPS

The last pair of parameters control filter rule "grouping". By default, all filter rules are placed in group 0 if no other group is specified. To add a rule to a non-default group, the group must first be started by creating a group *head*. If a packet matches a rule which is the *head* of a group, the filter processing then switches to the group, using that rule as the default for the group. If **quick** is used with a **head** rule, rule processing isn't stopped until it has returned from processing the group.

A rule may be both the head for a new group and a member of a non-default group (**head** and **group** may be used together in a rule).

head <n>
: indicates that a new group (number n) should be created.

group <n>
: indicates that the rule should be put in group (number n) rather than group 0.

LOGGING

When a packet is logged, with either the **log** action or option, the headers of the packet are written to the **ipl** packet logging pseudo-device. Immediately following the **log** keyword, the following qualifiers may be used (in order):

body indicates that the first 128 bytes of the packet contents will be logged after the headers.

first If log is being used in conjunction with a "keep" option, it is recommended that this option is also applied so that only the triggering packet is logged and not every packet which thereafter matches state information.

or-block
: indicates that, if for some reason the filter is unable to log the packet (such as the log reader being too slow) then the rule should be interpreted as if the action was **block** for this packet.

level <loglevel>
: indicates what logging facility and priority, or just priority with the default facility being used, will be used to log information about this packet using ipmon's -s option.

See ipl(4) for the format of records written to this device. The ipmon(8) program can be used to read and

format this log.

EXAMPLES

The **quick** option is good for rules such as:

```
block in quick from any to any with ipopts
```

which will match any packet with a non-standard header length (IP options present) and abort further processing of later rules, recording a match and also that the packet should be blocked.

The "fall-through" rule parsing allows for effects such as this:

```
block in from any to any port < 6000
pass in from any to any port >= 6000
block in from any to any port > 6003
```

which sets up the range 6000-6003 as being permitted and all others being denied. Note that the effect of the first rule is overridden by subsequent rules. Another (easier) way to do the same is:

```
block in from any to any port 6000 <> 6003
pass in from any to any port 5999 >< 6004
```

Note that both the "block" and "pass" are needed here to effect a result as a failed match on the "block" action does not imply a pass, only that the rule hasn't taken effect. To then allow ports < 1024, a rule such as:

```
pass in quick from any to any port < 1024
```

would be needed before the first block. To create a new group for processing all inbound packets on le0/le1/lo0, with the default being to block all inbound packets, we would do something like:

```
block in all
block in quick on le0 all head 100
block in quick on le1 all head 200
block in quick on lo0 all head 300
```

and to then allow ICMP packets in on le0, only, we would do:

```
pass in proto icmp all group 100
```

Note that because only inbound packets on le0 are used processed by group 100, there is no need to respecify the interface name. Likewise, we could further breakup processing of TCP, etc, as follows:

```
block in proto tcp all head 110 group 100
pass in from any to any port = 23 group 110
```

and so on. The last line, if written without the groups would be:

```
pass in on le0 proto tcp from any to any port = telnet
```

Note, that if we wanted to say "port = telnet", "proto tcp" would need to be specified as the parser interprets each rule on its own and qualifies all service/port names with the protocol specified.

FILES

/dev/ipauth
/dev/ipl
/dev/ipstate
/etc/hosts
/etc/services
/usr/share/examples/ipf Directory with examples.

SEE ALSO

ipftest(1), iptest(1), mkfilters(1), ipf(4), ipnat(5), ipf(8), ipfstat(8)

NAME

IP Filter

DESCRIPTION

IP Filter is a package providing packet filtering capabilities for a variety of operating systems. On a properly setup system, it can be used to build a firewall.

SEE ALSO

ipf(8), ipf(1), ipf(5), ipnat(8), ipnat(5), mkfilters(1)

NAME

ipmon, ipmon.conf − ipmon configuration file format

DESCRIPTION

The format for files accepted by ipmon is described by the following grammar:

```
"match" "{" matchlist "}" "do" "{" doing "}" ";"

matchlist ::= matching [ "," matching ] .
matching  ::= direction | dstip | dstport | every | group | interface |
              logtag | nattag | protocol | result | rule | srcip | srcport .

dolist ::= doing [ "," doing ] .
doing  ::= execute | save | syslog .

direction ::= "in" | "out" .
dstip     ::= "dstip" "=" ipv4 "/" number .
dstport   ::= "dstport" "=" number .
every     ::= "every" every-options .
execute   ::= "execute" "=" string .
group     ::= "group" "=" string | "group" "=" number .
interface ::= "interface" "=" string .
logtag    ::= "logtag" "=" string | "logtag" "=" number .
nattag    ::= "nattag" "=" string .
protocol  ::= "protocol" "=" string | "protocol" "=" number .
result    ::= "result" "=" result-option .
rule      ::= "rule" "=" number .
srcip     ::= "srcip" "=" ipv4 "/" number .
srcport   ::= "srcport" "=" number .
type      ::= "type" "=" ipftype .
ipv4      ::= number "." number "." number "." number .

every-options ::= "second" | number "seconds" | "packet" | number "packets" .
result-option ::= "pass" | "block" | "short" | "nomatch" | "log" .
ipftype ::= "ipf" | "nat" | "state" .
```

In addition, lines that start with a # are considered to be comments.

OVERVIEW

The ipmon configuration file is used for defining rules to be executed when logging records are read from **/dev/ipl.**

At present, only IPv4 matching is available for source/destination address matching.

MATCHING

Each rule for ipmon consists of two primary segments: the first describes how the log record is to be matched, the second defines what action to take if there is a positive match. All entries of the rules present in the file are compared for matches - there is no first or last rule match.

FILES

/dev/ipl
/dev/ipf
/dev/ipnat
/dev/ipstate
/etc/ipmon.conf

SEE ALSO

ipmon(8), ipl(4)

NAME

ipnat, ipnat.conf − IP NAT file format

DESCRIPTION

The format for files accepted by ipnat is described by the following grammar:

```
ipmap :: = mapblock | redir | map .

map ::= mapit ifname lhs "->" dstipmask [ mapicmp | mapport | mapproxy ]
      mapoptions .
mapblock ::= "map-block" ifname lhs "->" ipmask [ ports ] mapoptions .
redir ::= "rdr" ifname rlhs "->" ip [ "," ip ] rdrport rdroptions .

lhs ::= ipmask | fromto .
rlhs ::= ipmask dport | fromto .
dport ::= "port" portnum [ "-" portnum ] .
ports ::= "ports" numports | "auto" .
rdrport ::= "port" portnum .
mapit ::= "map" | "bimap" .
fromto ::= "from" object "to" object .
ipmask ::= ip "/" bits | ip "/" mask | ip "netmask" mask .
dstipmask ::= ipmask | "range" ip "-" ip .
mapicmp ::= "icmpidmap" "icmp" number ":" number .
mapport ::= "portmap" tcpudp portspec .
mapoptions ::= [ tcpudp ] [ "frag" ] [ age ] [ clamp ] .
rdroptions ::= rdrproto [ rr ] [ "frag" ] [ age ] [ clamp ] [ rdrproxy ] .

object :: = addr [ port-comp | port-range ] .
addr :: = "any" | nummask | host-name [ "mask" ipaddr | "mask" hexnumber ] .
port-comp :: = "port" compare port-num .
port-range :: = "port" port-num range port-num .
rdrproto ::= tcpudp | protocol .

rr ::= "round-robin" .
age ::= "age" decnumber [ "/" decnumber ] .
clamp ::= "mssclamp" decnumber .
tcpudp ::= "tcp/udp" | protocol .
mapproxy ::= "proxy" "port" port proxy-name ’/’ protocol
rdrproxy ::= "proxy" proxy-name .

protocol ::= protocol-name | decnumber .
nummask ::= host-name [ "/" decnumber ] .
portspec ::= "auto" | portnumber ":" portnumber .
port ::= portnumber | port-name .
portnumber ::= number { numbers } .
ifname ::= ’A’ - ’Z’ { ’A’ - ’Z’ } numbers .

numbers ::= ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’ .
```

For standard NAT functionality, a rule should start with **map** and then proceeds to specify the interface for which outgoing packets will have their source address rewritten.

Packets which will be rewritten can only be selected by matching the original source address. A netmask must be specified with the IP address.

The address selected for replacing the original is chosen from an IP#/netmask pair. A netmask of all 1’s indicating a hostname is valid. A netmask of 31 1’s (255.255.255.254) is considered invalid as there is no

space for allocating host IP#'s after consideration for broadcast and network addresses.

When remapping TCP and UDP packets, it is also possible to change the source port number. Either TCP or UDP or both can be selected by each rule, with a range of port numbers to remap into given as **port-number:port-number**.

COMMANDS

There are four commands recognised by IP Filter's NAT code:

map that is used for mapping one address or network to another in an unregulated round robin fashion;

rdr that is used for redirecting packets to one IP address and port pair to another;

bimap for setting up bidirectional NAT between an external IP address and an internal IP address and

map-block
which sets up static IP address based translation, based on a algorithm to squeeze the addresses to be translated into the destination range.

MATCHING

For basic NAT and redirection of packets, the address subject to change is used along with its protocol to check if a packet should be altered. The packet *matching* part of the rule is to the left of the "->" in each rule.

Matching of packets has now been extended to allow more complex compares. In place of the address which is to be translated, an IP address and port number comparison can be made using the same expressions available with **ipf**. A simple NAT rule could be written as:

map de0 10.1.0.0/16 -> 201.2.3.4/32

or as

map de0 from 10.1.0.0/16 to any -> 201.2.3.4/32

Only IP address and port numbers can be compared against. This is available with all NAT rules.

TRANSLATION

To the right of the "->" is the address and port specification which will be written into the packet providing it has already successfully matched the prior constraints. The case of redirections (**rdr**) is the simplest: the new destination address is that specified in the rule. For **map** rules, the destination address will be one for which the tuple combining the new source and destination is known to be unique. If the packet is either a TCP or UDP packet, the destination and source ports come into the equation too. If the tuple already exists, IP Filter will increment the port number first, within the available range specified with **portmap** and if there exists no unique tuple, the source address will be incremented within the specified netmask. If a unique tuple cannot be determined, then the packet will not be translated. The **map-block** is more limited in how it searches for a new, free and unique tuple, in that it will used an algorithm to determine what the new source address should be, along with the range of available ports - the IP address is never changed and nor does the port number ever exceed its allotted range.

ICMPIDMAP

ICMP messages can be divided into two groups: "errors" and "queries". ICMP errors are generated as a response of another IP packet. IP Filter will take care that ICMP errors that are the response of a NAT-ed IP packet are handled properly.

For 4 types of ICMP queries (echo request, timestamp request, information request and address mask request) IP Filter supports an additional mapping called "ICMP id mapping". All these 4 types of ICMP queries use a unique identifier called the ICMP id. This id is set by the process sending the ICMP query and it is usually equal to the process id. The receiver of the ICMP query will use the same id in its response, thus enabling the sender to recognize that the incoming ICMP reply is intended for him and is an answer to a query that he made. The "ICMP id mapping" feature modifies these ICMP id in a way identical to **portmap** for TCP or UDP.

The reason that you might want this, is that using this feature you don't need an IP address per host behind the NAT box, that wants to do ICMP queries. The two numbers behind the **icmpidmap** keyword are the

first and the last icmp id number that can be used. There is one important caveat: if you map to an IP address that belongs to the NAT box itself (notably if you have only a single public IP address), then you must ensure that the NAT box does not use the **icmpidmap** range that you specified in the **map** rule. Since the ICMP id is usually the process id, it is wise to restrict the largest permittable process id (PID) on your operating system to e.g. 63999 and use the range 64000:65535 for ICMP id mapping. Changing the maximal PID is system dependent. For most BSD derived systems can be done by changing PID_MAX in /usr/include/sys/proc.h and then rebuild the system.

KERNEL PROXIES

IP Filter comes with a few, simple, proxies built into the code that is loaded into the kernel to allow secondary channels to be opened without forcing the packets through a user program. The current state of the proxies is listed below, as one of three states:

Aging - protocol is roughly understood from the time at which the proxy was written but it is not well tested or maintained;

Developmental - basic functionality exists, works most of the time but may be problematic in extended real use;

Experimental - rough support for the protocol at best, may or may not work as testing has been at best sporadic, possible large scale changes to the code in order to properly support the protocol.

Mature - well tested, protocol is properly understood by the proxy;

The currently compiled in proxy list is as follows:

FTP - Mature

IRC - Experimental

rpcbind - Experimental

H.323 - Experimental

Real Audio (PNA) - Aging

IPsec - Developmental

netbios - Experimental

R-command - Mature

TRANSPARENT PROXIES

True transparent proxying should be performed using the redirect (**rdr**) rules directing ports to localhost (127.0.0.1) with the proxy program doing a lookup through **/dev/ipnat** to determine the real source and address of the connection.

LOAD-BALANCING

Two options for use with **rdr** are available to support primitive, *round-robin* based load balancing. The first option allows for a **rdr** to specify a second destination, as follows:

rdr le0 203.1.2.3/32 port 80 -> 203.1.2.3,203.1.2.4 port 80 tcp

This would send alternate connections to either 203.1.2.3 or 203.1.2.4. In scenarios where the load is being spread amongst a larger set of servers, you can use:

rdr le0 203.1.2.3/32 port 80 -> 203.1.2.3,203.1.2.4 port 80 tcp round-robin
rdr le0 203.1.2.3/32 port 80 -> 203.1.2.5 port 80 tcp round-robin

In this case, a connection will be redirected to 203.1.2.3, then 203.1.2.4 and then 203.1.2.5 before going back to 203.1.2.3. In accomplishing this, the rule is removed from the top of the list and added to the end, automatically, as required. This will not effect the display of rules using "ipnat -l", only the internal application order.

EXAMPLES

This section deals with the **map** command and its variations.

To change IP#'s used internally from network 10 into an ISP provided 8 bit subnet at 209.1.2.0 through the ppp0 interface, the following would be used:

map ppp0 10.0.0.0/8 -> 209.1.2.0/24

The obvious problem here is we're trying to squeeze over 16,000,000 IP addresses into a 254 address space. To increase the scope, remapping for TCP and/or UDP, port remapping can be used;

map ppp0 10.0.0.0/8 -> 209.1.2.0/24 portmap tcp/udp 1025:65000

which falls only 527,566 'addresses' short of the space available in network 10. If we were to combine these rules, they would need to be specified as follows:

map ppp0 10.0.0.0/8 -> 209.1.2.0/24 portmap tcp/udp 1025:65000
map ppp0 10.0.0.0/8 -> 209.1.2.0/24

so that all TCP/UDP packets were port mapped and only other protocols, such as ICMP, only have their IP# changed. In some instances, it is more appropriate to use the keyword **auto** in place of an actual range of port numbers if you want to guarantee simultaneous access to all within the given range. However, in the above case, it would default to 1 port per IP address, since we need to squeeze 24 bits of address space into 8. A good example of how this is used might be:

map ppp0 172.192.0.0/16 -> 209.1.2.0/24 portmap tcp/udp auto

which would result in each IP address being given a small range of ports to use (252). In all cases, the new port number that is used is deterministic. That is, port X will always map to port Y. WARNING: It is not advisable to use the **auto** feature if you are map'ing to a /32 (i.e. 0/32) because the NAT code will try to map multiple hosts to the same port number, outgoing and ultimately this will only succeed for one of them. The problem here is that the **map** directive tells the NAT code to use the next address/port pair available for an outgoing connection, resulting in no easily discernible relation between external addresses/ports and internal ones. This is overcome by using **map-block** as follows:

map-block ppp0 172.192.0.0/16 -> 209.1.2.0/24 ports auto

For example, this would result in 172.192.0.0/24 being mapped to 209.1.2.0/32 with each address, from 172.192.0.0 to 172.192.0.255 having 252 ports of its own. As opposed to the above use of **map**, if for some reason the user of (say) 172.192.0.2 wanted 260 simultaneous connections going out, they would be limited to 252 with **map-block** but would just *move on* to the next IP address with the **map** command.

FILES

/dev/ipnat
/etc/services
/etc/hosts
/usr/share/examples/ipf Directory with examples.

SEE ALSO

ipnat(4), hosts(5), ipf(5), services(5), ipf(8), ipnat(8)

NAME

ippool, ippool.conf − IP Pool file format

DESCRIPTION

The format for files accepted by ippool is described by the following grammar:

```
line ::= table | groupmap .
table ::= "table" role tabletype .
groupmap ::= "group-map" inout role number ipfgroup
tabletype ::= ipftree | ipfhash .

role ::= "role" "=" "ipf" .
inout ::= "in" | "out" .

ipftree ::= "type" "=" "tree" number "{" addrlist "}" .
ipfhash ::= "type" "=" "hash" number hashopts "{" hashlist "}" .

ipfgroup ::= setgroup hashopts "{" grouplist "}" |
        hashopts "{" setgrouplist "}" .
setgroup ::= "group" "=" groupname .

hashopts ::= size [ seed ] | seed .

size ::= "size" number .
seed ::= "seed" number .

addrlist ::= [ "!" ] addrmask ";" [ addrlist ] .
grouplist ::= groupentry ";" [ grouplist ] | addrmask ";" [ grouplist ] .

setgrouplist ::= groupentry ";" [ setgrouplist ] .

groupentry ::= addrmask "," setgroup .

hashlist ::= hashentry ";" [ hashlist ] .
hashentry ::= addrmask .

addrmask ::= ipaddr | ipaddr "/" mask .

mask ::= number | ipaddr .

groupname ::= number | name .

number ::= digit { digit } .

ipaddr  = host-num "." host-num "." host-num "." host-num .
host-num = digit [ digit [ digit ] ] .

digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .
name ::= letter { letter | digit } .
```

The IP pool configuration file is used for defining a single object that contains a reference to multiple IP address/netmask pairs. A pool may consist of a mixture of netmask sizes, from 0 to 32.

At this point in time, only IPv4 addressing is supported.

OVERVIEW

The IP pool configuration file provides for defining two different mechanisms for improving speed in matching IP addresses with rules. The first, **table** , defines a lookup *table* to provide a single reference in a filter rule to multiple targets and the second, **group-map** , provides a mechanism to target multiple groups from a single filter line.

The **group-map** command can only be used with filter rules that use the **call** command to invoke either **fr_srcgrpmap** or **fr_dstgrpmap** , to use the source or destination address, respectively, for determining which filter group to jump to next for continuation of filter packet processing.

POOL TYPES

Two storage formats are provided: hash tables and tree structure. The hash table is intended for use with objects all containing the same netmask or a few different sized netmasks of non-overlapping address space and the tree is designed for being able to support exceptions to a covering mask, in addition to normal searching as you would do with a table. It is not possible to use the tree data storage type with **group-map** configuration entries.

POOL ROLES

When a pool is defined in the configuration file, it must have an associated role. At present the only supported role is **ipf.** Future development will see futher expansion of their use by other sections of IPFilter code.

EXAMPLES

The following examples show how the pool configuration file is used with the ipf configuration file to enhance the ability for the ipf configuration file to be succinct in meaning.

1 The first example shows how a filter rule makes reference to a specific pool for matching of the source address.
pass in from pool/100 to any

The pool configuration, which matches IP addresses 1.1.1.1 and any in 2.2.0.0/16, except for those in 2.2.2.0/24.

```
table role = ipf type = tree number = 100
    { 1.1.1.1/32; 2.2.0.0/16; !2.2.2.0/24 };
```

2 The following ipf.conf extract uses the fr_srcgrpmap/fr_dstgrpmap lookups to use the **group-map** facility to lookup the next group to use for filter processing, providing the **call** filter rule is matched.

```
call now fr_srcgrpmap/1010 in all
call now fr_dstgrpmap/2010 out all
pass in all group 1020
block in all group 1030
pass out all group 2020
block out all group 2040
```

A ippool configuration to work with the above ipf.conf file might look like this:

```
group-map in role = ipf number = 1010
        { 1.1.1.1/32, group = 1020; 3.3.0.0/16, group = 1030; };
group-map out role = ipf number = 2010 group = 2020
        { 2.2.2.2/32; 4.4.0.0/16; 5.0.0.0/8, group = 2040; };
```

FILES

/dev/iplookup
/etc/ippool.conf
/etc/hosts

SEE ALSO

ippool(8), hosts(5), ipf(5), ipf(8), ipnat(8)

NAME

ipscan, ipscan.conf − ipscan file format

DESCRIPTION

WARNING: This feature is to be considered experimental and may change significantly until a final implementation is drawn up.

The format for files accept by ipscan currently follow this rough grammar:

```
line     ::= name ":" matchup [ "," matchup ] "=" action .
matchup  ::= "(" ")" | "(" literal ")" | "(" literal "," match ")" .
action   ::= result | result "else" result .
result   ::= "close" | "track" | redirect .
redirect ::= "redirect" ip-address [ "(" "," port-number ")" ] .
match    ::= { match-char }
match-char ::= "*" | "?" | "."
```

In this example an ip-address is a dotted-quad IPv4 address and a port-number is a number betwee 1 and 65535, inclusive. The match string is must be of same length as the literal string that it is matching (literal). The length of either string is limited to 16 bytes.

Currently, the redirect option is not yet been implemented.

```
#
# * = match any character, . = exact match, ? = case insensitive
#
# Scan for anything that looks like HTTP and redirect it to the local
# proxy.  One catch - this feature (redirect) is not yet implemented.
#
http : ("GET ", "???." ) = redirect(127.0.0.1)
#
# Track ssh connections (i.e do nothing)
#
ssh : (), ("SSH-") = track
#
# Things which look like smtp to be tracked else closed.
# Client can start with EHLO (ESMTP) or HELO (SMTP).
#
smtp : ("HELO ", "**??."), ("220 ", "....") = track else close
#
```

FILES

/etc/ipscan.conf

SEE ALSO

ipscan(8)

NAME

ipsec.conf — static IPsec configuration read at system startup

DESCRIPTION

The **ipsec.conf** file is read at system startup time if **ipsec** is set to "yes" in rc.conf(5). setkey(8) is run with the **-f** option to load in IPsec manual keys and policies from /etc/ipsec.conf at boot time, before any interfaces are configured.

Please see the setkey(8) manpage for all the commands available.

FILES

/etc/ipsec.conf The file **ipsec.conf** resides in /etc.

SEE ALSO

ipsec(4), setkey(8)

HISTORY

The **ipsec.conf** file appeared in NetBSD 1.5.

NAME

ipsend − IP packet description language

DESCRIPTION

The **ipsend** program expects, with the **-L** option, input to be a text file which fits the grammar described below. The purpose of this grammar is to allow IP packets to be described in an arbitary way which also allows encapsulation to be so done to an arbitary level.

GRAMMAR

```
line ::= iface | arp | send | defrouter | ipv4line .

iface ::= ifhdr "{" ifaceopts "}" ";" .
ifhdr ::= "interface" | "iface" .
ifaceopts ::= "ifname" name | "mtu" mtu | "v4addr" ipaddr |
              "eaddr" eaddr .

send ::= "send" ";" | "send" "{" sendbodyopts "}" ";" .
sendbodyopts ::= sendbody [ sendbodyopts ] .
sendbody ::= "ifname" name | "via" ipaddr .

defrouter ::= "router" ipaddr .

arp ::= "arp" "{" arpbodyopts "}" ";" .
arpbodyopts ::= arpbody [ arpbodyopts ] .
arpbody ::= "v4addr" ipaddr | "eaddr" eaddr .

bodyline ::= ipv4line | tcpline | udpline | icmpline | dataline .

ipv4line ::= "ipv4" "{" ipv4bodyopts "}" ";" .
ipv4bodyopts ::= ipv4body [ ipv4bodyopts ] | bodyline .
ipv4body ::= "proto" protocol | "src" ipaddr | "dst" ipaddr |
             "off" number | "v" number | "hl" number| "id" number |
             "ttl" number | "tos" number | "sum" number | "len" number |
             "opt" "{" ipv4optlist "}" ";" .
ipv4optlist ::= ipv4option [ ipv4optlist ] .
ipv4optlist = "nop" | "rr" | "zsu" | "mtup" | "mtur" | "encode" | "ts" |
              "tr" | "sec" | "lsrr" | "e-sec" | "cipso" | "satid" |
              "ssrr" | "addext" | "visa" | "imitd" | "eip" | "finn" |
              "secclass" ipv4secclass.
ipv4secclass := "unclass" | "confid" | "reserv-1" | "reserv-2" |
                "reserv-3" | "reserv-4" | "secret" | "topsecret" .

tcpline ::= "tcp" "{" tcpbodyopts "}" ";" .
tcpbodyopts ::= tcpbody [ tcpbodyopts ] | bodyline .
tcpbody ::= "sport" port | "dport" port | "seq" number | "ack" number |
            "off" number | "urp" number | "win" number | "sum" number |
            "flags" tcpflags | data .

udpline ::= "udp" "{" udpbodyopts "}" ";" .
udpbodyopts ::= udpbody [ udpbodyopts ] | bodyline .
udpbody ::= "sport" port | "dport" port | "len" number | "sum" number |
            data .

icmpline ::= "icmp" "{" icmpbodyopts "}" ";" .
icmpbodyopts ::= icmpbody [ icmpbodyopts ] | bodyline .
icmpbody ::= "type" icmptype [ "code" icmpcode ] .
```

```
icmptype ::= "echorep" | "echorep" "{" echoopts "}" ";" | "unreach" |
             "unreach" "{" unreachtype "}" ";" | "squench" | "redir" |
             "redir" "{" redirtype "}" ";" | "echo" "{" echoopts "}" ";" |
             "echo" | "routerad" | "routersol" | "timex" |
             "timex" "{" timextype "}" ";" | "paramprob" |
             "paramprob" "{" parapptype "}" ";" | "timest" | "timestrep" |
             "inforeq" | "inforep" | "maskreq" | "maskrep" .

echoopts ::= echoopts [ icmpechoopts ] .
unreachtype ::= "net-unr" | "host-unr" | "proto-unr" | "port-unr" |
             "needfrag" | "srcfail" | "net-unk" | "host-unk" | "isolate" |
             "net-prohib" | "host-prohib" | "net-tos" | "host-tos" |
             "filter-prohib" | "host-preced" | "cutoff-preced" .
redirtype ::= "net-redir" | "host-redir" | "tos-net-redir" |
              "tos-host-redir" .
timextype ::= "intrans" | "reass" .
paramptype ::= "optabsent" .

data       ::= "data" "{" databodyopts "}" ";" .
databodyopts ::= "len" number | "value" string | "file" filename .

icmpechoopts ::= "icmpseq" number | "icmpid" number .
```

COMMANDS

Before sending any packets or defining any packets, it is necessary to describe the interface(s) which will be used to send packets out.

interface
: is used to describe a network interface. The description included need not match the actual configuration currently employed by the operating system.

send
: is used to actually send out a packet across the network. If the destination is not specified, it will attempt to send the packet directly out on the network to the destination without routing it.

router
: configures the default router for ipsend, as distinct from the default route installed in the kernel.

ipv4
: is used to describe an IP (version 4) packet. IP header fields can be specified, including options, followed by a data section which may contain further protocol headers.

IPv4

hl <number>
: manually specifies the IP header length (automatically adjusts with the presence of IP options and defaults to 5);

v <number>
: set the IP version. Default is 4.

tos <number>
: set the type of service (TOS) field in the IP header. Default is 0.

len <number>
: manually specifies the length of the IP packet. The length will automatically be adjusted to accommodate data or further protocol headers.

off <number>
: sets the fragment offset field of the IP packet. Default is 0.

ttl <number>
: sets the time to live (TTL) field of the IP header. Default is 60.

proto <protocol>
sets the protocol field of the IP header. The protocol can either be a number or a name found in **/etc/protocols**.

sum manually specifies the checksum for the IP header. If left unset (0), it will be calculated prior to being sent.

src manually specifies the source address of the IP header. If left unset, it will default to the host's IP address.

dst sets the destination of the IP packet. The default is 0.0.0.0.

opt is used to include IP options in the IP header.

tcp is used to indicate the a TCP protocol header is to follow. See the **TCP** section for TCP header options.

udp is used to indicate the a UDP protocol header is to follow. See the **UDP** section for UDP header options.

icmp is used to indicate the a ICMP protocol header is to follow. See the **ICMP** section for ICMP header options.

data is used to indicate that raw data is to be included in the IP packet. See the **DATA** section for details on options available.

IPv4 Options

these keywords indicate that the relevant IP option should be added to the IP header (the header length field will be adjusted appropriately).

nop No Operation [RFC 791] (space filler).

rr <number>
Record Router [RFC 791]. The number given specifies the number of **bytes** to be used for storage. This should be a multiple of 4 for proper operation.

zsu Experimental Measurement.

mtup [RFC 1191].
MTU Probe.

mtur [RFC 1191].
MTU Ready.

encode

ts Timestamp [RFC 791].

tr Traceroute [RFC 1393].

sec-class <security-level>, sec
Security [RFC 1108]. This option specifies the security label for the packet. Using **sec** sets up the framework of the security option but unless **sec-class** is given, the level may not be set.

lsrr <ip-address>
Loose Source Route [RFC 791].

e-sec Extended Security [RFC 1108].

cipso Commercial Security.

satid Stream ID [RFC 791].

ssrr <ip-address>
Strict Source Route [RFC 791].

addext Address Extension

visa Experimental Access Control.

imitd IMI Traffic Descriptor.

eip [RFC 1358].

finn Experimental Flow Control.

TCP

sport <port>
sets the source port to the number/name given. Default is 0.

dport <port>
sets the destination port to the number/name given. Default is 0.

seq <number>
sets the sequence number to the number specified. Default is 0.

ack <number>
sets the acknowledge number to the number specified. Default is 0.

off <number>
sets the offset value for the start of data to the number specified. This implies the size of the TCP header. It is automatically adjusted if TCP options are included and defaults to 5.

urp <number>
sets the value of the urgent data pointer to the number specified. Default is 0.

win <number>
sets the size of the TCP window to the number specified. Default is 4096.

sum <number>
manually specifies the checksum for the TCP pseudo-header and data. If left unset, it defaults to 0 and is automatically calculated.

flags <tcp-flags>
sets the TCP flags field to match the flags specified. Valid flags are "S" (SYN), "A" (ACK), "R" (RST), "F" (FIN), "U" (URG), "P" (PUSH).

opt indicates that TCP header options follow. As TCP options are added to the TCP header, the **off** field is updated to match.

data indicates that a data section is to follow and is to be included as raw data, being appended to the header.

TCP options

With a TCP header, it is possible to append a number of header options. The TCP header offset will be updated automatically to reflect the change in size. The valid options are: **nop** No Operation, **eol** End Of (option) List, **mss [size]** Maximum Segment Size - this sets the maximum receivable size of a packet containing data, **wscale** Window Scale, **ts** Timestamp.

UDP

sport <port>
sets the source port to the number/name given. Default is 0.

dport <port>
sets the destination port to the number/name given. Default is 0.

len <number>
manually specifies the length of the UDP header and data. If left unset, it is automatically adjusted to match the header presence and any data if present.

sum <number>
manually specifies the checksum for the UDP pseudo-header and data. If left unset, it defaults to 0 and is automatically calculated.

data indicates that a data section is to follow and is to be included as raw data, being appended to the header.

ICMP

type <icmptype>
sets the ICMP type according the to the icmptype tag. This may either be a number or one of the recognised tags (see the **ICMP TYPES** section for a list of names recognised).

code <icmpcode>
sets the ICMP code.

data indicates that a data section is to follow and is to be included as raw data, being appended to the header.

DATA

Each of the following extend the packet in a different way. **Len** just increases the length (without adding any content), **value** uses a string and **file** a file.

len <number>
extend the length of the packet by **number** bytes (without filling those bytes with any particular data).

value <string>
indicates that the string provided should be added to the current packet as data. A string may be a consecutive list of characters and numbers (with no white spaces) or bounded by "'s (may not contain them, even if \'d). The \ character is recognised with the appropriate C escaped values, including octal numbers.

file <filename>
reads data in from the specified file and appends it to the current packet. If the new total length would exceed 64k, an error will be reported.

ICMP TYPES

echorep
Echo Reply.

unreach [unreachable-code]
Generic Unreachable error. This is used to indicate that an error has occurred whilst trying to send the packet across the network and that the destination cannot be reached. The unreachable code names are: **net-unr** network unreachable, **host-unr** host unreachable, **proto-unr** protocol unreachable, **port-unr** port unreachable, **needfrag**, **srcfail** source route failed, **net-unk** network unknown, **host-unk** host unknown, **isolate**, **net-prohib** administratively prohibited contact with network, **host-prohib** administratively prohibited contact with host, **net-tos** network unreachable with given TOS, **host-tos** host unreachable with given TOS, **filter-prohib** packet prohibited by packet filter, **host-preced**, **cutoff-preced**.

squench
Source Quence.

redir [redirect-code]
Redirect (routing). This is used to indicate that the route being chosen for forwarding the packet is suboptimal and that the sender of the packet should be routing packets via another route. The redirect code names are: **net-redir** redirect packets for a network, **host-redir** redirect packets for a host, **tos-net-redir** redirect packets for a network with a given TOS, **tos-host-redir** redirect packets for a host with a given TOS.

echo Echo.

routerad
Router Advertisement.

routersol
Router solicitation.

timex [timexceed-code]
Time Exceeded. This is used to indicate that the packet failed to reach the destination because it was in transit too long (i.e. ttl reached 0). The valid code names are: **intrans**, **reass** could not reassemble packet from fragments within a given time.

paramprob [paramprob-code]
Parameter problem. There is only one available parameter problem code name: **optabsent**.

timest Time stamp request.

timestrep [{ timestamp-code }]
Time stamp reply. In a timestamp reply, it is possible to supply the following values: **rtime**, **otime**, **ttime**.

inforeq Information request.

inforep Information reply.

maskreq
Address mask request.

maskrep
Address mask reply.

FILES

/etc/hosts
/etc/protocols
/etc/services

SEE ALSO

ipsend(1), iptest(1), hosts(5), protocols(5), services(5)

NAME

isdnd.acct — isdn4bsd ISDN management daemon accounting file format

DESCRIPTION

The file isdnd.acct contains accounting information which is written if the variable *useacctfile* in the isdnd(8) configuration file isdnd.rc(5) is set to *on* and charging information transmission has been subscribed for the ISDN connection (AOCD or AOCE).

If the variable *acctall* is set to *on*, accounting information is written even if the local site was not charged or no charging information is available or is not subscribed.

The general format of an accounting line is as follows:

```
FROM - UNTIL NAME UNITS (SECONDS) (INBYTES/OUTBYTES)
```

FROM is the time the connection was established in the format

```
Day.Month.Year Hour:Minutes:seconds
```

UNTIL is the time the connection was closed. The format is the same as described for *FROM* above.

NAME is the symbolic name got from the *name* entry of the isdnd.rc(5) config file for this connection.

UNITS is the amount of charging units billed for the connection.

SECONDS is the number of seconds the connection lasted.

INBYTES and *OUTBYTES* is the (optional) number of bytes that were transferred.

FILES

/var/log/isdnd.acct The default accounting information file for the **isdnd** ISDN daemon.

EXAMPLES

This is a typical accounting line:

```
12.06.97 10:41:37 - 12.06.97 10:45:18 GROGGY 2 (65) (4711/1147)
```

SEE ALSO

isdnd.rc(5), isdnd(8)

AUTHORS

The isdnd(8) daemon and this manual page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

isdnd.rates — isdn4bsd ISDN management daemon rates description file

DESCRIPTION

The file isdnd.rates contains descriptions how long charging units last at a given time of day, day of week and the distance to the destination. If this file is available, this information may be used by the isdnd(8) ISDN connection management daemon to calculate the short hold time for a connection.

The format of a rate entry line is as follows:

The first field, the (*rate-code*) defines a collection of rates (for each day of the week) which can be referenced in the isdnd(8) configuration file isdnd.rc(5). This field must start with the identifier "ra" followed by a digit in the range of zero to four.

The second field, the (*day-number*) selects the day of week for which this entry defines the rates, where 0 stands for Sunday, 1 for Monday and so on until the digit 6 which stands for Saturday.

The rest of the line consists of one or more space separated fields which have the following syntax:

start_hour.start_minutes-end_hour.end_minutes:charge_unit_length

Start_hour and start_minutes define the begin of a time section and end_hour and end_minutes define the end. Charge_unit_length define the length of a charging unit in the previously defined time section. No spaces or tabs are allowed inside this field. The hour and minutes specifications MUST have exactly 2 digits, in case just one digit is needed, a leading 0 must be used.

For example,

14.00-18.00:90

defines, that between 2:00 PM and 6:00 PM the length of one charging unit lasts 90 seconds.

FILES

/etc/isdn/isdnd.rates The default rates specification file for the **isdnd** ISDN daemon.

EXAMPLES

The line:

```
ra0 0 00.00-05.00:240 05.00-21.00:150 21.00-24.00:240
```

defines the unit lengths for a Sunday.

SEE ALSO

isdnd.rc(5), isdnd(8)

AUTHORS

The rates subsystem for the isdnd(8) daemon to which **isdnd.rates** belongs was designed and written by Gary Jennejohn.

The isdnd(8) daemon and this manual page were written by
Hellmuth Michaelis ⟨hm@kts.org⟩.

NAME

isdnd.rc — isdn4bsd ISDN connection management daemon config file format

DESCRIPTION

The file /etc/isdn/isdnd.rc contains (if not otherwise specified on the command line) the runtime configuration for the isdnd(8) ISDN connection management daemon which is part of the isdn4bsd package.

The configuration file consists of keywords which start in column 1 followed by one or more spaces or tabs, an equal sign, one or more spaces or tabs and a keyword dependent parameter value.

A line beginning with '#' is treated as a comment line.

For keywords requiring the specification of a boolean value, the truth value can be either *yes* or *on* while the false value can be either *no* or *off*.

The configuration file consists of one *system* section, one or more optional *controller* sections and one or more *entry* sections. In the *system* section parameters regarding the daemon operation or parameters not associated with a single remote connection can be set. In the *controller* section parameters regarding a particular controller can be set. In the *entry* section(s) parameters directly associated with a single remote connection can be set.

The following keywords are recognized by **isdnd**:

system This keyword starts the system configuration section. It must not have a parameter and may be used only once. The keyword is mandatory. The following keywords are valid in the system configuration section:

- acctall — If this parameter is set to *on*, accounting information is written even if the local site was not charged or no charging information is available or is not subscribed. (optional)
- acctfile — Specifies the name of the accounting file which is used when the keyword *useacctfile* (see below) is set to *on*. See also system keyword *rotatesuffix*. If this keyword is omitted the system default is used. (optional)
- aliasing — If this parameter is set to *on*, alias processing of telephone-number to name is enabled (see also the *aliasfile* keyword below). The default is off. (optional)
- aliasfile — Specifies the name of the telephone number-to-name alias database file shared with the isdntel(8) utility when alias processing is enabled via the *aliasing* keyword. (optional)
- beepconnect — In full-screen mode, if this parameter is set to *on*, ring the bell when connecting or disconnecting a call.
- extcallattr — If this parameter is set to *on*, the extended caller attributes "screening indicator" and "presentation indicator" are written to the log-file. The default is off. (optional)
- holidayfile — Specifies the name of the holiday file containing the dates of holidays. This file is used in conjunction with the *valid* keyword to lookup the dates of holidays. (optional)
- isdntime — If this parameter is set to *on*, date/time information from the exchange (if provided) is written to the log-file. The default is off. (optional)
- mailer — This keyword is used to specify the path/name of a mail program which which is able to use the "-s" flag to specify a subject on its command line. In case of a fatal error exit of **isdnd.rc** this program is used to send mail to an administrator specified by the keyword *mailto*. (optional)

`mailto` This keyword is used to specify the email address of someone to notify in case of a fatal error exit of **`isdnd.rc`**. (See also keyword *mailer*). (optional)

`monitor-allowed`
If this parameter is set to *on* or *yes*, monitoring via a local or remote machine is enabled. This parameter is optional and is set to *off* by default.

`monitor-port`
sets the TCP port number for remote monitoring. This integer parameter is optional and is set to port 451 by default.

`monitor` This keyword specifies a local socket name or a host or network for remote monitoring. The *monitor* specification may either be:

`the name of a local (UNIX-domain) socket`
this MUST start with a "/", example: /var/run/isdn-monitor

`a dotted-quad host specification`
example: 192.168.1.2

`a dotted-quad network address with netmask`
example: 192.168.1.0/24

`a resolvable host name`
example: localhost

`a resolvable network name with netmask`
example: up-vision-net/24

`monitor-access`
This keyword specifies the access rights for a previously used *monitor* keyword. The supported access rights are:

`fullcmd`
`restrictedcmd`
`channelstate`
`logevents`
`callin`
`callout`

`ratesfile` Specifies the name of the ratesfile. If this keyword is omitted the system default is used. (optional)

`regexpr` This keyword is used to specify regular expressions. It can be specified more than once up to a compile time dependent value (currently set to 5 by the MAX_RE definition in the source).

All specified regular expressions are compared to the log strings at runtime and if a match is found, a program is run with the log text as a parameter (see also the keyword *regprog* below).

For an explanation how regular expressions are specified, please have a look at `re_format`(7) and `regex`(3). The *extended* regular expression syntax is supported here.

Hint: it might be necessary to properly quote the expression to avoid improper interpretation by the configuration file parser. (optional)

`regprog` This keyword is used to specify the name of a program which is run in case a corresponding regular expression is matched by a logging string. **`Isdnd`** expects to find the program below the path `/etc/isdn` which is prepended to the string specified as a parameter to this keyword. (optional)

`rotatesuffix`
Specifies a suffix for renaming the log- and the accounting-filename. In case rotatesuffix is used and a USR1 signal is sent to isdnd, the log-file and the

accounting file is not only closed and reopened but the old log-file is also renamed to the former filename with the rotatesuffix string appended. If this keyword is omitted, the log-files are just closed and reopened; this is also the default behavior. (optional)

`useacctfile` If this parameter is set to *on* charging (if available) and accounting information is written to the accounting file. (optional)

`controller`

This keyword starts the controller configuration section. It must not have a parameter and may be used once for every controller. The keyword is optional. The following keywords are valid in a controller configuration section:

`firmware` This keyword is used to specify the path of the firmware file that will be loaded to the card once **`isdnd`** is started. This keyword is useful with active ISDN cards.

`protocol` This keyword is used to set the D-channel protocol for the S0-bus a controller is connected to. The following parameters are currently supported:

`dss1` The DSS1 or so-called "Euro-ISDN" D-channel protocol according to ITU Recommendations Q.921 and Q.931.

`d64s` An ISDN leased line with a single B-channel (called D64S in Germany).

`entry` This keyword starts one configuration entry. It must not have a parameter. This keyword must be used at least once. The following keywords are valid in an entry section:

`answerprog` This keyword is used to specify the name of a program which is run in case an incoming telephone connection specified *answer* in its configuration entry. The default name is *answer*. **`Isdnd`** expects to find this program beneath the path `/etc/isdn` which is prepended to the string specified as a parameter to this keyword. (optional)

`alert` is used to specify a time in seconds to wait before accepting a call. This keyword is only usable for incoming telephone calls (dialin-reaction = answer). It is used to have a chance to accept an incoming call on the phone before the answering machine starts to run. The minimum value for the alert parameter is 5 seconds and the maximum parameter allowed is 180 seconds. (optional)

`autoupdown` For network interfaces using ISDN as a transport medium it does not make sense to mark the interfaces UP before running **`isdnd`**. Typically these interfaces are configured, but marked down, in the respective `ifconfig.*` file. When starting, **`isdnd`** recognizes these interfaces (configured with some address, marked down, and having a matching config entry) and marks them up. On shutdown, **`isdnd`** marks all interfaces changed at startup DOWN again.

In rare circumstances you might not want this automatic handling. In this cases add an *autoupdown=no* line to the config file entry.

`b1protocol` The B channel layer 1 protocol used for this connection. The keyword is mandatory. The currently configurable values are:

`hdlc` HDLC framing.

`raw` No framing at all (used for telephony).

`budget-calloutperiod`

is used to specify a time period in seconds. Within this period, the number of calls specified by *budget-calloutncalls* are allowed to succeed, any further attempt to call out will be blocked for the rest of the time left in the time period. (optional)

`budget-calloutncalls`
: The number of outgoing calls allowed within the time period specified by *budget-calloutperiod*. (optional)

`budget-calloutsfile`
: A path/filename to which the number of successful callouts are written. The contents of the file is preserved when it exists during startup of isdnd. The format of this file is: start time, last update time, number of calls. (optional)

`budget-calloutsfile-rotate`
: If set to *on* rotate budget-calloutsfile every night when an attempt is made to update the file on a new day. The statistics for the previous day are written to a file with the filename specified by budget-calloutsfile to which a hyphen and the new day's (!) day of month number is appended. (optional)

`budget-callbackperiod`
`budget-callbackncalls`
`budget-callbacksfile`
`budget-calloutsfile-rotate`
: See *budget-calloutperiod*, *budget-calloutncalls and budget-calloutsfile budget-calloutsfile-rotate* above. These are used to specify the budgets for calling back a remote site.

`callbackwait` The time in seconds to wait between hanging up the call from a remote site and calling back the remote site. (optional)

`calledbackwait`
: The time in seconds to wait for a remote site calling back the local site after a call from the local site to the remote site has been made. (optional)

`connectprog` specifies a program run every time after a connection is established and address negotiation is complete (i.e.: the connection is usable). **`Isdnd`** expects to find the program below the path `/etc/isdn` which is prepended to the string specified as a parameter to this keyword. The programs specified by connect and disconnect will get the following command line arguments: -d (device) -f (flag) [-a (addr)] where *device* is the name of device, e.g. "ippp0", *flag* will be "up" if connection just got up, or "down" if interface changed to down state and *addr* the address that got assigned to the interface as a dotted-quad IP address (optional, only if it can be figured out by isdnd). (optional)

`dialin-reaction`
: Used to specify what to do when an incoming connection request is received. The keyword is mandatory. The currently supported parameters are:

`accept`	Accept an incoming call.
`reject`	Reject an incoming call.
`ignore`	Ignore an incoming call.
`answer`	Start telephone answering for an incoming voice call.
`callback`	When a remote site calls, hang up and call back the remote site.

`dialout-type` This keyword is used to configure what type of dialout mode is used. The keyword is mandatory. The currently supported parameters are:

`normal`
: Normal behavior, call the remote site which is supposed to accept the call.

`calledback`
: Callback behavior, call the remote side which rejects the call and calls us back.

`dialrandincr` When dialing or re-dialing and this parameter is set to *on*, the dial retry time is added with a random value (currently 0...3 seconds) to minimize the chance of two sites dialing synchronously so each gets a busy each time it dials because the other side is also dialing.

`dialretries` The number of dialing retries before giving up. Setting this to *-1* gives an unlimited number of retries! (optional)

`direction` This keyword is used to configure if incoming and outgoing, incoming-only or outgoing only connections are possible. The keyword is optional, the default is *inout*.

The currently supported parameters are:

`inout` Normal behavior, connection establishment is possible from remote and local.

`in` Only incoming connections are possible.

`out` Only outgoing connections are possible.

`disconnectprog`
specifies a program run every time after a connection was shut down. **Isdnd** expects to find the program below the path `/etc/isdn` which is prepended to the string specified as a parameter to this keyword. (optional)

`downtries` is used to configure the number of unsuccessful tries (= retry cycles!) before the interface is disabled (for *downtime* seconds). (see also the keyword *usedown* further up). This keyword is optional.

`downtime` is used to configure the time in seconds an interface is disabled after the configured number of *downtries*. (see also the keyword *usedown* further up). This keyword is optional and is set to 60 seconds by default.

`earlyhangup` A (safety) time in seconds which specifies the time to hang up before an expected next charging unit will occur. (optional)

`idle-algorithm-outgoing`
The algorithm used to determine when to hang up an outgoing call when the line becomes idle. The current algorithms are:

`fix-unit-size`
idle algorithm which assumes fixed sized changing units during the whole call.

`var-unit-size`
idle algorithm which assumes that the charging is time based after the first units time has expired.

`idletime-outgoing`
The time in seconds an outgoing connection must be idle before hanging up. An idle timeout of zero disables this functionality. (optional)

`idletime-incoming`
The time in seconds an incoming connection must be idle before hanging up. An idle timeout of zero disables this functionality. (optional)

`isdncontroller`
The ISDN controller number to be used for connections for this entry. (mandatory)

`isdnchannel` The ISDN controller channel number to be used for connections for this entry. In case a channel is explicitly selected here, the SETUP message will request this channel but mark the request as *preferred* (the indicated channel is preferred) instead of exclusive (only the indicated channel is acceptable). Thus the exchange is still free to select another than the requested channel! (mandatory)

`isdntxdel-incoming`
How long to delay the transmission of the first packet after a successful connection is made for *incoming* ISDN connections. The specification unit is 1/100 second. A zero (0) disables this feature and is the default value. This feature is implemented (and makes sense only) for the `irip`(4) IP over raw HDLC ISDN driver. (optional)

`isdntxdel-outgoing`
How long to delay the transmission of the first packet after a successful connection is made for *outgoing* ISDN connections. The specification unit is 1/100 second. A zero (0) disables this feature and is the default value. This feature is implemented (and makes sense only) for the `irip`(4) IP over raw HDLC ISDN driver. (optional)

`local-phone-dialout`
The local telephone number used when the local site dials out. When dialing out to a remote site, the number specified here is put into the *Calling Party Number Information Element*.

This keyword is mandatory for the *irip* user-land interfaces.

`local-phone-incoming`
The local telephone number used for verifying the destination of incoming calls. When a remote site dials in, this number is used to verify that it is the local site which the remote site wants to connect to. It is compared with the *Called Party Number Information Element* got from the telephone exchange.

This keyword is mandatory for the *irip* interfaces.

`name` Defines a symbolic name for this configuration entry. Its purpose is to use this name in the full-screen display for easy identification of a link to a remote site and for accounting purposes. (mandatory)

`ppp-auth-paranoid`
If set to *no*, the remote site is not required to prove its authenticity for connections that are initiated by the local site. The default is *yes* and requires the remote site to always authenticate.

This keyword is only used if *ppp-send-auth* has been set to pap or chap for an *ippp* PPP interface. (optional)

`ppp-auth-rechallenge`
Set to *no*, if the other side does not support re-challenging for chap. The default is *yes*, which causes verification of the remote site's authenticity once in a while.

This keyword is only used if *ppp-expect-auth* has been set to chap for an *ippp* PPP interface. (optional)

`ppp-expect-auth`
The local site expects the authenticity of the remote site to be proved by the specified method. The supported methods are:

- *none* Do not require the other side to authenticate. Typical uses are dial-out to an ISP (many ISPs do not authenticate themselves to clients) or offering anonymous dial-in at the local site.
- *chap* The preferred authentication method, which does not require a password to be sent in the clear.
- *pap* The unprotected authentication method, which allows anybody watching the wire to grab name and password.

If *ppp-auth-paranoid* is set to *no* (the default is *yes*) outgoing connections will not require the remote site to authenticate itself.

This keyword is only used for the *ippp* PPP interfaces. (optional)

`ppp-expect-name`
The name that has to be provided by the remote site to prove its authenticity.

This keyword is only used if *ppp-expect-auth* has been set to pap or chap for an *ippp* PPP interface. (optional)

`ppp-expect-password`
The secret that has to be provided by the remote site to prove its authenticity.

This keyword is only used if *ppp-expect-auth* has been set to pap or chap for an *ippp* PPP interface. (optional)

`ppp-send-auth` The authentication method required by the remote site. The currently supported parameters are:

- *none* The remote site does not expect or support authentication.
- *chap* The preferred authentication method, which does not require a password to be sent in the clear.
- *pap* The unprotected authentication method, which allows anybody watching the wire to grab name and password.

This keyword is only used for the *ippp* PPP interfaces. (optional)

`ppp-send-name` The authentication name sent to the remote site.

This keyword is only used if *ppp-send-auth* has been set to pap or chap for an *ippp* PPP interface. (optional)

`ppp-send-password`
The secret used to prove the local site's authenticity to the remote site.

This keyword is only used if *ppp-send-auth* has been set to pap or chap for an *ippp* PPP interface. (optional)

`ratetype` The rate entry used from the rates file. (optional)
For example, ratetype=0 selects lines beginning "ra0" in /etc/isdn/isdnd.rates; (typically ra0 lines are a set of tables for local call rates on different days of the week & times per day).

`recoverytime` The time in seconds to wait between dial retries. (optional)

`remdial-handling`
is used to specify the dialout behavior in case more than one outgoing number is specified. The currently supported parameters are:

- *first* For every new (non-retry) call setup, start with the first number.
- *last* For every new (non-retry) call setup, start with the last number with which a successful connection was made.
- *next* For every new (non-retry) call setup, start with the next number which follows the last one used.

`remote-phone-dialout`
The remote telephone number used when the local site dials out. When dialing out to a remote site, the number specified here is put into the *Called Party Number Information Element*.

This keyword is mandatory for the *irip* interfaces. It may be specified more than once to try to dial to several numbers until one succeeds.

`remote-phone-incoming`
The remote telephone number used to verify an incoming call. When a remote site dials in, this number is used to verify that it is the correct remote site which is herewith authorized to connect into the local system. This parameter is compared against the *Calling Party Number Information Element* got from the telephone exchange.

This keyword is mandatory for the irip interfaces.

This keyword may have a wildcard parameter '*' to permit anyone dialing in.

`unitlength` The length of a charging unit in seconds. This is used in conjunction with the idletime to decide when to hang up a connection. (optional)

`unitlengthsrc` This keyword is used to specify from which source `isdnd`(8) takes the unitlength for short-hold mode. The currently configurable values are:

- *`none`* Then unitlength is not specified anywhere.
- *`cmdl`* Use the unitlength specified on the command line.
- *`conf`* Use the unitlength specified in the configuration file with the keyword *unitlength*.
- *`rate`* Use the unitlength from the ratesfile specified in the configuration file with the keyword *ratetype*.
- *`aocd`* Use a dynamically calculated unitlength in case AOCD is subscribed on the ISDN line. (AOCD is an acronym for ''Advice Of Charge During the call'' which is a service provided by the telecommunications (ie phone) provider, to indicate billable units).

`usrdevicename` Specifies the user-land interface which is used for interfacing ISDN B channel data to the user-land. The keyword is mandatory. This keyword accepts the following parameters:

- *`irip`* This parameter configures a raw HDLC IP over ISDN interface.
- *`ippp`* This parameter configures a synchronous PPP over ISDN interface.
- *`rbch`* This specifies a Raw B Channel access interface.
- *`isdntel`* ISDN telephony.
- *`ing`* configures a ISDN B-channel to NetGraph interface.

`usrdeviceunit` Specifies the unit number for the device which is specified with usrdevicename.

`usedown` is used to enable the use of the keywords *downtries* and *downtime* in the entries section(s). It is used in the **`isdnd`** daemon to dynamically enable and disable the IP interfaces to avoid excessive dialing activities in case of transient failures (such as busy lines). This parameter is optional and is set to *off* by default.

`valid` *Note*: this feature is considered experimental! The parameter to this keyword is a string specifying a time range within which this entry is valid. The time specification consists of a list of weekdays and/or a holiday indicator (see also the *holidayfile* keyword in the system section) separated by commas followed by an optional daytime range specification in the form hh:mm-hh:mm. The weekdays are specified as numbers from 0 to 6 and the number 7 for holidays:

- *`0`* Sunday
- *`1`* Monday

2 Tuesday
3 Wednesday
4 Thursday
5 Friday
6 Saturday
7 a Holiday

The following examples describe the "T-ISDN xxl" tariff of the german Telekom:

1,2,3,4,5,6,09:00-18:00
Monday through Saturday, daytime 9:00 to 18:00

1,2,3,4,5,6,18:00-9:00
Monday through Saturday, nighttime 18:00 to 9:00

0,7 Sunday and on holidays, all 24 hours

The use of this keyword is optional.

IDLETIME CALCULATION AND SHORT-HOLD MODE

incoming calls

It is assumed that the calling side knows most about charging structures and such and as a consequence only the keyword *idletime-incoming* has a function for incoming calls.

For incoming calls the line is constantly monitored, and in case there was not traffic taking place for the time in seconds specified by *idletime-incoming* the call is closed.

Typically, *idletime-incoming* is used as a last resort and is therefore set much higher than a charging unit time: typical values are one to five minutes.

outgoing calls

Outgoing call disconnect time can be set up in one of three ways:

simple mode

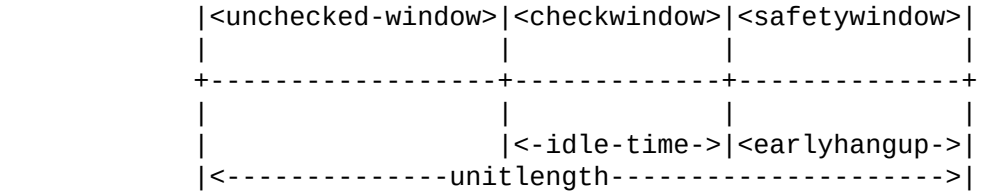
For simple mode, the *idle-algorithm-outgoing* must be *fix-unit-size* and the selected *unitlength* must be 0 (zero) and *idletime-outgoing* greater zero.

The outgoing traffic is constantly monitored, and in case there was not traffic taking place for the time in seconds specified by *idletime-outgoing* the call is closed.

Typical values in simple mode are 10 to 30 seconds.

shorthold mode for fixed unit charging

For shorthold mode, the *idle-algorithm-outgoing*
must be *fix-unit-size*
and the selected *unitlength* and *idletime-outgoing* must be greater than 0 (zero); *earlyhangup* must be ≥ 0 (zero).



During the unchecked window which is (unitlength - (idle-time+earlyhangup)) in length, no idle check is done. After the unchecked window has ended, the line is checked for idle-time length if no traffic takes place. In case there was traffic detected in the check-window, the same procedure is restarted at the beginning of the next unit. In case no traffic was detected during the check-window, the line is closed at the end of the check window.

Notice: *unitlength* must (!) be greater than the sum of *idletime-outgoing* and *earlyhangup*!

`shorthold mode for variable unit charging`

For shorthold mode, the *idle-algorithm-outgoing* must be *var-unit-size* and the selected *unitlength* and *idletime-outgoing* must be greater than 0 (zero);

This shorthold mode is suitable when your calls are billed on the elapse time of the call plus a fixed connection charge. For example British Telecom bill this way.

Each call is divided into two periods, the first is the *unchecked* period and the second is the *checked*. The *checked* period starts 1 second before the first units time expires.

During the *checked* period if there is no traffic for *idle-time* seconds the call is disconnected.

```
|<---unchecked------------------>|<------checked------>
+------------------+-------------+
|                  |<-idle-time->|
|<--------------unitlength------->|
```

Experience shows that useful values for idle-time are from 15 to 30 seconds.

If idle-time is too short an application that is not yet finished with the network will cause a new call to be placed.

FILES

`/etc/isdn/isdnd.rc` The default configuration file for the **isdnd** ISDN daemon.

SEE ALSO

`regex`(3), `re_format`(7), `isdnd`(8), `isdnmonitor`(8)

AUTHORS

The `isdnd`(8) daemon and this manual page were written by Hellmuth Michaelis ⟨hm@kts.org⟩.

Additions to this manual page by
Barry Scott ⟨barry@scottb.demon.co.uk⟩.

NAME

krb5.conf — configuration file for Kerberos 5

SYNOPSIS

```
#include <krb5/krb5.h>
```

DESCRIPTION

The **krb5.conf** file specifies several configuration parameters for the Kerberos 5 library, as well as for some programs.

The file consists of one or more sections, containing a number of bindings. The value of each binding can be either a string or a list of other bindings. The grammar looks like:

```
file:
        /* empty */
        sections

sections:
        section sections
        section

section:
        '[' section_name ']' bindings

section_name:
        STRING

bindings:
        binding bindings
        binding

binding:
        name '=' STRING
        name '=' '{' bindings '}'

name:
        STRING
```

STRINGs consists of one or more non-whitespace characters.

STRINGs that are specified later in this man-page uses the following notation.

boolean
: values can be either yes/true or no/false.

time
: values can be a list of year, month, day, hour, min, second. Example: 1 month 2 days 30 min. If no unit is given, seconds is assumed.

etypes
: valid encryption types are: des-cbc-crc, des-cbc-md4, des-cbc-md5, des3-cbc-sha1, arcfour-hmac-md5, aes128-cts-hmac-sha1-96, and aes256-cts-hmac-sha1-96 .

address
: an address can be either a IPv4 or a IPv6 address.

Currently recognised sections and bindings are:

`[appdefaults]`

Specifies the default values to be used for Kerberos applications. You can specify defaults per application, realm, or a combination of these. The preference order is:

1. *application realm option*
2. *application option*
3. *realm option*
4. *option*

The supported options are:

`forwardable =` *boolean*
When obtaining initial credentials, make the credentials forwardable.

`proxiable =` *boolean*
When obtaining initial credentials, make the credentials proxiable.

`no-addresses =` *boolean*
When obtaining initial credentials, request them for an empty set of addresses, making the tickets valid from any address.

`ticket_lifetime =` *time*
Default ticket lifetime.

`renew_lifetime =` *time*
Default renewable ticket lifetime.

`encrypt =` *boolean*
Use encryption, when available.

`forward =` *boolean*
Forward credentials to remote host (for `rsh`(1), `telnet`(1), etc).

`[libdefaults]`

`default_realm =` *REALM*
Default realm to use, this is also known as your "local realm". The default is the result of **`krb5_get_host_realm`**(*`local hostname`*).

`clockskew =` *time*
Maximum time differential (in seconds) allowed when comparing times. Default is 300 seconds (five minutes).

`kdc_timeout =` *time*
Maximum time to wait for a reply from the kdc, default is 3 seconds.

`v4_name_convert`

`v4_instance_resolve`
These are described in the `krb5_425_conv_principal`(3) manual page.

`capath = {`

destination-realm `=` *next-hop-realm*

…

`}`
This is deprecated, see the `capaths` section below.

`default_cc_name` = *ccname*
: the default credentials cache name. The string can contain variables that are expanded on runtime. Only support variable now is `%{uid}` that expands to the current user id.

`default_etypes` = *etypes* ...
: A list of default encryption types to use.

`default_etypes_des` = *etypes* ...
: A list of default encryption types to use when requesting a DES credential.

`default_keytab_name` = *keytab*
: The keytab to use if no other is specified, default is “FILE:/etc/krb5.keytab”.

`dns_lookup_kdc` = *boolean*
: Use DNS SRV records to lookup KDC services location.

`dns_lookup_realm` = *boolean*
: Use DNS TXT records to lookup domain to realm mappings.

`kdc_timesync` = *boolean*
: Try to keep track of the time differential between the local machine and the KDC, and then compensate for that when issuing requests.

`max_retries` = *number*
: The max number of times to try to contact each KDC.

`large_msg_size` = *number*
: The threshold where protocols with tiny maximum message sizes are not considered usable to send messages to the KDC.

`ticket_lifetime` = *time*
: Default ticket lifetime.

`renew_lifetime` = *time*
: Default renewable ticket lifetime.

`forwardable` = *boolean*
: When obtaining initial credentials, make the credentials forwardable. This option is also valid in the [realms] section.

`proxiable` = *boolean*
: When obtaining initial credentials, make the credentials proxiable. This option is also valid in the [realms] section.

`verify_ap_req_nofail` = *boolean*
: If enabled, failure to verify credentials against a local key is a fatal error. The application has to be able to read the corresponding service key for this to work. Some applications, like `su`(1), enable this option unconditionally.

`warn_pwexpire` = *time*
: How soon to warn for expiring password. Default is seven days.

`http_proxy` = *proxy-spec*
: A HTTP-proxy to use when talking to the KDC via HTTP.

`dns_proxy` = *proxy-spec*
: Enable using DNS via HTTP.

`extra_addresses =` *address ...*
A list of addresses to get tickets for along with all local addresses.

`time_format =` *string*
How to print time strings in logs, this string is passed to `strftime`(3).

`date_format =` *string*
How to print date strings in logs, this string is passed to `strftime`(3).

`log_utc =` *boolean*
Write log-entries using UTC instead of your local time zone.

`scan_interfaces =` *boolean*
Scan all network interfaces for addresses, as opposed to simply using the address associated with the system's host name.

`fcache_version =` *int*
Use file credential cache format version specified.

`krb4_get_tickets =` *boolean*
Also get Kerberos 4 tickets in **`kinit`**, **`login`**, and other programs. This option is also valid in the [realms] section.

`fcc-mit-ticketflags =` *boolean*
Use MIT compatible format for file credential cache. It's the field ticketflags that is stored in reverse bit order for older than Heimdal 0.7. Setting this flag to `TRUE` make it store the MIT way, this is default for Heimdal 0.7.

`[domain_realm]`
This is a list of mappings from DNS domain to Kerberos realm. Each binding in this section looks like:

```
domain = realm
```

The domain can be either a full name of a host or a trailing component, in the latter case the domain-string should start with a period. The trailing component only matches hosts that are in the same domain, ie ".example.com" matches "foo.example.com", but not "foo.test.example.com".

The realm may be the token 'dns_locate', in which case the actual realm will be determined using DNS (independently of the setting of the 'dns_lookup_realm' option).

`[realms]`

REALM `= {`

`kdc =` *[service/]host[:port]*
Specifies a list of kdcs for this realm. If the optional *port* is absent, the default value for the "kerberos/udp" "kerberos/tcp", and "http/tcp" port (depending on service) will be used. The kdcs will be used in the order that they are specified.

The optional *service* specifies over what medium the kdc should be contacted. Possible services are "udp", "tcp", and "http". Http can also be written as "http://". Default service is "udp" and "tcp".

`admin_server =` *host[:port]*
Specifies the admin server for this realm, where all the modifications to the database are performed.

`kpasswd_server = host[:port]`
Points to the server where all the password changes are performed. If there is no such entry, the kpasswd port on the admin_server host will be tried.

`krb524_server = host[:port]`
Points to the server that does 524 conversions. If it is not mentioned, the krb524 port on the kdcs will be tried.

`v4_instance_convert`

`v4_name_convert`

`default_domain`
See `krb5_425_conv_principal`(3).

`tgs_require_subkey`
a boolan variable that defaults to false. Old DCE secd (pre 1.1) might need this to be true.

`}`

`[capaths]`

client-realm `= {`

server-realm `=` *hop-realm* ...
This serves two purposes. First the first listed *hop-realm* tells a client which realm it should contact in order to ultimately obtain credentials for a service in the *server-realm*. Secondly, it tells the KDC (and other servers) which realms are allowed in a multi-hop traversal from *client-realm* to *server-realm*. Except for the client case, the order of the realms are not important.

}

`[logging]`

entity `=` *destination*
Specifies that *entity* should use the specified `destination` for logging. See the `krb5_openlog`(3) manual page for a list of defined destinations.

`[kdc]`

`database = {`

`dbname =` *DATABASENAME*
Use this database for this realm. See the info documetation how to configure diffrent database backends.

`realm =` *REALM*
Specifies the realm that will be stored in this database. It realm isn't set, it will used as the default database, there can only be one entry that doesn't have a `realm` stanza.

`mkey_file = FILENAME`
Use this keytab file for the master key of this database. If not specified *DATABASENAME*.mkey will be used.

`acl_file = PA FILENAME`
Use this file for the ACL list of this database.

`log_file = FILENAME`
Use this file as the log of changes performed to the database. This file is used by **`ipropd-master`** for propagating changes to slaves.

`}`

`max-request =` *SIZE*
Maximum size of a kdc request.

`require-preauth =` *BOOL*
If set pre-authentication is required. Since krb4 requests are not pre-authenticated they will be rejected.

`ports =` *list of ports*
List of ports the kdc should listen to.

`addresses =` *list of interfaces*
List of addresses the kdc should bind to.

`enable-kerberos4 =` *BOOL*
Turn on Kerberos 4 support.

`v4-realm =` *REALM*
To what realm v4 requests should be mapped.

`enable-524 =` *BOOL*
Should the Kerberos 524 converting facility be turned on. Default is the same as *enable-kerberos4*.

`enable-http =` *BOOL*
Should the kdc answer kdc-requests over http.

`enable-kaserver =` *BOOL*
If this kdc should emulate the AFS kaserver.

`check-ticket-addresses =` *BOOL*
Verify the addresses in the tickets used in tgs requests.

`allow-null-ticket-addresses =` *BOOL*
Allow address-less tickets.

`allow-anonymous =` *BOOL*
If the kdc is allowed to hand out anonymous tickets.

`encode_as_rep_as_tgs_rep =` *BOOL*
Encode as-rep as tgs-rep tobe compatible with mistakes older DCE secd did.

`kdc_warn_pwexpire =` *TIME*
The time before expiration that the user should be warned that her password is about to expire.

`logging =` *Logging*
What type of logging the kdc should use, see also [logging]/kdc.

`use_2b = {`

principal = *BOOL*
boolean value if the 524 daemon should return AFS 2b tokens for *principal*.

...

}

hdb-ldap-structural-object *structural object*
If the LDAP backend is used for storing principals, this is the structural object that will be used when creating and when reading objects. The default value is account .

hdb-ldap-create-base *creation dn*
is the dn that will be appended to the principal when creating entries. Default value is the search dn.

[kadmin]

require-preauth = *BOOL*
If pre-authentication is required to talk to the kadmin server.

password_lifetime = *time*
If a principal already have its password set for expiration, this is the time it will be valid for after a change.

default_keys = *keytypes...*
For each entry in *default_keys* try to parse it as a sequence of *etype:salttype:salt* syntax of this if something like:

[(des|des3|etype):](pw-salt|afs3-salt)[:string]

If *etype* is omitted it means everything, and if string is omitted it means the default salt string (for that principal and encryption type). Additional special values of keytypes are:

v5 The Kerberos 5 salt *pw-salt*

v4 The Kerberos 4 salt *des:pw-salt:*

use_v4_salt = *BOOL*
When true, this is the same as

default_keys = des3:pw-salt v4

and is only left for backwards compatibility.

[password-quality]
Check the Password quality assurance in the info documentation for more information.

check_library = *library-name*
Library name that contains the password check_function

check_function = *function-name*
Function name for checking passwords in check_library

policy_libraries = *library1 ... libraryN*
List of libraries that can do password policy checks

policies = *policy1 ... policyN*
List of policy names to apply to the password. Builtin policies are among other minimum-length, character-class, external-check.

ENVIRONMENT

`KRB5_CONFIG` points to the configuration file to read.

FILES

`/etc/krb5.conf` configuration file for Kerberos 5.

EXAMPLES

```
[libdefaults]
        default_realm = FOO.SE
[domain_realm]
        .foo.se = FOO.SE
        .bar.se = FOO.SE
[realms]
        FOO.SE = {
                kdc = kerberos.foo.se
                v4_name_convert = {
                        rcmd = host
                }
                v4_instance_convert = {
                        xyz = xyz.bar.se
                }
                default_domain = foo.se
        }
[logging]
        kdc = FILE:/var/heimdal/kdc.log
        kdc = SYSLOG:INFO
        default = SYSLOG:INFO:USER
```

DIAGNOSTICS

Since **`krb5.conf`** is read and parsed by the krb5 library, there is not a lot of opportunities for programs to report parsing errors in any useful format. To help overcome this problem, there is a program **`verify_krb5_conf`** that reads **`krb5.conf`** and tries to emit useful diagnostics from parsing errors. Note that this program does not have any way of knowing what options are actually used and thus cannot warn about unknown or misspelled ones.

SEE ALSO

`kinit`(1), `krb5_425_conv_principal`(3), `krb5_openlog`(3), `strftime`(3), `verify_krb5_conf`(8)

NAME

ld.so.conf — run-time link-editor configuration file

DESCRIPTION

The **ld.so.conf** file specifies additional default directories (beyond the standard set, normally "/usr/lib").

On a.out(5) systems, this file is scanned by ldconfig(8) to create the hints files used by the run-time linker /usr/libexec/ld.so to locate shared libraries.

On elf(5) systems, this file is scanned directly by the run-time linker /usr/libexec/ld.elf_so.

Lines beginning with "#" are treated as comments and ignored. Any other non-blank lines beginning with '/' are stripped of leading whitespace and trailing comments (introduced with "#") together with any preceding whitespace, then treated as directories to be scanned for shared libraries to add to the hints.

On elf(5) lines that do not begin with a '/' are parsed as hardware dependent per library directives:

<library_name> <sysctl_variable> <variable_value>[,...]:<library_name>[,...] ...

If there is no match, the standard action is taken.

FILES

/etc/ld.so.conf

EXAMPLES

libm.so.0 machdep.fpu_present 1:libm387.so.0,libm.so.0

The above line loads both libm387 and libm when the sysctl(3) variable fpu_present has the value 1.

SEE ALSO

ld.aout_so(1), ld.elf_so(1), a.out(5), elf(5), ldconfig(8)

HISTORY

The **ld.so.conf** file appeared in NetBSD 1.3. The ELF support for it was added in NetBSD 1.5.

BUGS

Directory names containing the comment character ("#") and/or leading or trailing whitespace cannot be included. (Embedded blanks are allowed, however.)

NAME

ldap.conf, .ldaprc − ldap configuration file

SYNOPSIS

ETCDIR/ldap.conf, .ldaprc

DESCRIPTION

If the environment variable **LDAPNOINIT** is defined, all defaulting is disabled.

The *ldap.conf* configuration file is used to set system-wide defaults to be applied when running *ldap* clients.

Users may create an optional configuration file, *ldaprc* or *.ldaprc*, in their home directory which will be used to override the system-wide defaults file. The file *ldaprc* in the current working directory is also used.

Additional configuration files can be specified using the **LDAPCONF** and **LDAPRC** environment variables. **LDAPCONF** may be set to the path of a configuration file. This path can be absolute or relative to the current working directory. The **LDAPRC**, if defined, should be the basename of a file in the current working directory or in the user's home directory.

Environmental variables may also be used to augment the file based defaults. The name of the variable is the option name with an added prefix of **LDAP**. For example, to define **BASE** via the environment, set the variable **LDAPBASE** to the desired value.

Some options are user−only. Such options are ignored if present in the *ldap.conf* (or file specified by **LDAPCONF**).

OPTIONS

The configuration options are case-insensitive; their value, on a case by case basis, may be case-sensitive.

Blank lines and lines beginning with a hash mark ('#') are ignored up to their end.

Valid lines are made of an option's name (a sequence of non-blanks, conventionally written in uppercase, although not required), followed by a value. The value starts with the first non-blank character after the option's name, and terminates at the end of the line, or at the last sequence of blanks before the end of the line. The tokenization of the value, if any, is delegated to the handler(s) for that option, if any. Quoting values that contain blanks may be incorrect, as the quotes would become part of the value. For example,

```
URI     "ldap:// ldaps://"
```

is incorrect, while

```
URI     ldap:// ldaps://
```

is correct (note the absence of the double quotes).

A line cannot be longer than LINE_MAX, which should be more than 2000 bytes on all platforms. There is no mechanism to split a long line on multiple lines, either for beautification or to overcome the above limit.

The different configuration options are:

URI <ldap[si]://[name[:port]] ...>

Specifies the URI(s) of an LDAP server(s) to which the *LDAP* library should connect. The URI scheme may be any of **ldap**, **ldaps** or **ldapi**, which refer to LDAP over TCP, LDAP over SSL (TLS) and LDAP over IPC (UNIX domain sockets), respectively. Each server's name can be specified as a domain-style name or an IP address literal. Optionally, the server's name can followed by a ':' and the port number the LDAP server is listening on. If no port number is provided, the default port for the scheme is used (389 for ldap://, 636 for ldaps://). For LDAP over IPC, **name** is the name of the socket, and no **port** is required, nor allowed; note that directory separators must be URL-encoded, like any other characters that are special to URLs; so the socket

/usr/local/var/ldapi

must be specified as

ldapi://%2Fusr%2Flocal%2Fvar%2Fldapi

A space separated list of URIs may be provided.

BASE <base>
Specifies the default base DN to use when performing ldap operations. The base must be specified as a Distinguished Name in LDAP format.

BINDDN <dn>
Specifies the default bind DN to use when performing ldap operations. The bind DN must be specified as a Distinguished Name in LDAP format. **This is a user−only option.**

DEREF <when>
Specifies how alias dereferencing is done when performing a search. The **<when>** can be specified as one of the following keywords:

never Aliases are never dereferenced. This is the default.

searching
Aliases are dereferenced in subordinates of the base object, but not in locating the base object of the search.

finding Aliases are only dereferenced when locating the base object of the search.

always Aliases are dereferenced both in searching and in locating the base object of the search.

HOST <name[:port] ...>
Specifies the name(s) of an LDAP server(s) to which the *LDAP* library should connect. Each server's name can be specified as a domain-style name or an IP address and optionally followed by a ':' and the port number the ldap server is listening on. A space separated list of hosts may be provided. **HOST** is deprecated in favor of **URI**.

NETWORK_TIMEOUT <integer>
Specifies the timeout (in seconds) after which the poll(2)/select(2) following a connect(2) returns in case of no activity.

PORT <port>
Specifies the default port used when connecting to LDAP servers(s). The port may be specified as a number. **PORT** is deprecated in favor of **URI.**

REFERRALS <on/true/yes/off/false/no>
Specifies if the client should automatically follow referrals returned by LDAP servers. The default is on. Note that the command line tools **ldapsearch**(1) &co always override this option.

SIZELIMIT <integer>
Specifies a size limit to use when performing searches. The number should be a non-negative integer. *SIZELIMIT* of zero (0) specifies unlimited search size.

TIMELIMIT <integer>
Specifies a time limit to use when performing searches. The number should be a non-negative integer. *TIMELIMIT* of zero (0) specifies unlimited search time to be used. **VERSION {2|3}** Specifies what version of the LDAP protocol should be used.

TIMEOUT <integer>
Specifies a timeout (in seconds) after which calls to synchronous LDAP APIs will abort if no response is received. Also used for any **ldap_result**(3) calls where a NULL timeout parameter is supplied.

SASL OPTIONS

If OpenLDAP is built with Simple Authentication and Security Layer support, there are more options you can specify.

SASL_MECH <mechanism>
Specifies the SASL mechanism to use. **This is a user−only option.**

SASL_REALM <realm>
Specifies the SASL realm. **This is a user−only option.**

SASL_AUTHCID <authcid>
Specifies the authentication identity. **This is a user−only option.**

SASL_AUTHZID <authcid>
Specifies the proxy authorization identity. **This is a user−only option.**

SASL_SECPROPS <properties>
Specifies Cyrus SASL security properties. The **<properties>** can be specified as a comma-separated list of the following:

none (without any other properties) causes the properties defaults ("noanonymous,noplain") to be cleared.

noplain
disables mechanisms susceptible to simple passive attacks.

noactive
disables mechanisms susceptible to active attacks.

nodict disables mechanisms susceptible to passive dictionary attacks.

noanonymous
disables mechanisms which support anonymous login.

forwardsec
requires forward secrecy between sessions.

passcred
requires mechanisms which pass client credentials (and allows mechanisms which can pass credentials to do so).

minssf=<factor>
specifies the minimum acceptable *security strength factor* as an integer approximating the effective key length used for encryption. 0 (zero) implies no protection, 1 implies integrity protection only, 56 allows DES or other weak ciphers, 112 allows triple DES and other strong ciphers, 128 allows RC4, Blowfish and other modern strong ciphers. The default is 0.

maxssf=<factor>
specifies the maximum acceptable *security strength factor* as an integer (see **minssf** description). The default is **INT_MAX**.

maxbufsize=<factor>
specifies the maximum security layer receive buffer size allowed. 0 disables security layers. The default is 65536.

TLS OPTIONS

If OpenLDAP is built with Transport Layer Security support, there are more options you can specify. These options are used when an **ldaps:// URI** is selected (by default or otherwise) or when the application negotiates TLS by issuing the LDAP StartTLS operation.

TLS_CACERT <filename>
Specifies the file that contains certificates for all of the Certificate Authorities the client will recognize.

TLS_CACERTDIR <path>
Specifies the path of a directory that contains Certificate Authority certificates in separate individual files. The **TLS_CACERT** is always used before **TLS_CACERTDIR.** This parameter is ignored with GNUtls.

TLS_CERT <filename>
Specifies the file that contains the client certificate. **This is a user−only option.**

TLS_KEY <filename>
Specifies the file that contains the private key that matches the certificate stored in the **TLS_CERT** file. Currently, the private key must not be protected with a password, so it is of critical importance that the key file is protected carefully. **This is a user−only option.**

TLS_CIPHER_SUITE <cipher-suite-spec>
Specifies acceptable cipher suite and preference order. <cipher-suite-spec> should be a cipher specification for OpenSSL, e.g., HIGH:MEDIUM:+SSLv2.

TLS_RANDFILE <filename>
Specifies the file to obtain random bits from when /dev/[u]random is not available. Generally set to the name of the EGD/PRNGD socket. The environment variable RANDFILE can also be used to specify the filename. This parameter is ignored with GNUtls.

TLS_REQCERT <level>
Specifies what checks to perform on server certificates in a TLS session, if any. The **<level>** can be specified as one of the following keywords:

never The client will not request or check any server certificate.

allow The server certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, it will be ignored and the session proceeds normally.

try The server certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, the session is immediately terminated.

demand | hard
These keywords are equivalent. The server certificate is requested. If no certificate is provided, or a bad certificate is provided, the session is immediately terminated. This is the default setting.

TLS_CRLCHECK <level>
Specifies if the Certificate Revocation List (CRL) of the CA should be used to verify if the server certificates have not been revoked. This requires **TLS_CACERTDIR** parameter to be set. This parameter is ignored with GNUtls. **<level>** can be specified as one of the following keywords:

none No CRL checks are performed

peer Check the CRL of the peer certificate

all Check the CRL for a whole certificate chain

TLS_CRLFILE <filename>
Specifies the file containing a Certificate Revocation List to be used to verify if the server certificates have not been revoked. This parameter is only supported with GNUtls.

ENVIRONMENT VARIABLES

LDAPNOINIT
disable all defaulting

LDAPCONF
: path of a configuration file

LDAPRC
: basename of ldaprc file in $HOME or $CWD

LDAP<option-name>
: Set <option-name> as from ldap.conf

FILES

ETCDIR/ldap.conf
: system-wide ldap configuration file

$HOME/ldaprc, $HOME/.ldaprc
: user ldap configuration file

$CWD/ldaprc
: local ldap configuration file

SEE ALSO

ldap(3), **ldap_set_option**(3), **ldap_result**(3), **openssl**(1), **sasl**(3)

AUTHOR

Kurt Zeilenga, The OpenLDAP Project

ACKNOWLEDGEMENTS

NAME

ldap.conf, .ldaprc − ldap configuration file

SYNOPSIS

/etc/openldap/ldap.conf, .ldaprc

DESCRIPTION

If the environment variable **LDAPNOINIT** is defined, all defaulting is disabled.

The *ldap.conf* configuration file is used to set system-wide defaults to be applied when running *ldap* clients.

Users may create an optional configuration file, *ldaprc* or *.ldaprc*, in their home directory which will be used to override the system-wide defaults file. The file *ldaprc* in the current working directory is also used.

Additional configuration files can be specified using the **LDAPCONF** and **LDAPRC** environment variables. **LDAPCONF** may be set to the path of a configuration file. This path can be absolute or relative to the current working directory. The **LDAPRC**, if defined, should be the basename of a file in the current working directory or in the user's home directory.

Environmental variables may also be used to augment the file based defaults. The name of the variable is the option name with an added prefix of **LDAP**. For example, to define **BASE** via the environment, set the variable **LDAPBASE** to the desired value.

Some options are user−only. Such options are ignored if present in the *ldap.conf* (or file specified by **LDAPCONF**).

OPTIONS

The configuration options are case-insensitive; their value, on a case by case basis, may be case-sensitive.

Blank lines and lines beginning with a hash mark ('#') are ignored up to their end.

Valid lines are made of an option's name (a sequence of non-blanks, conventionally written in uppercase, although not required), followed by a value. The value starts with the first non-blank character after the option's name, and terminates at the end of the line, or at the last sequence of blanks before the end of the line. The tokenization of the value, if any, is delegated to the handler(s) for that option, if any. Quoting values that contain blanks may be incorrect, as the quotes would become part of the value. For example,

```
URI     "ldap:// ldaps://"
```

is incorrect, while

```
URI     ldap:// ldaps://
```

is correct (note the absence of the double quotes).

A line cannot be longer than LINE_MAX, which should be more than 2000 bytes on all platforms. There is no mechanism to split a long line on multiple lines, either for beautification or to overcome the above limit.

The different configuration options are:

URI <ldap[si]://[name[:port]] ...>

Specifies the URI(s) of an LDAP server(s) to which the *LDAP* library should connect. The URI scheme may be any of **ldap**, **ldaps** or **ldapi**, which refer to LDAP over TCP, LDAP over SSL (TLS) and LDAP over IPC (UNIX domain sockets), respectively. Each server's name can be specified as a domain-style name or an IP address literal. Optionally, the server's name can followed by a ':' and the port number the LDAP server is listening on. If no port number is provided, the default port for the scheme is used (389 for ldap://, 636 for ldaps://). For LDAP over IPC, **name** is the name of the socket, and no **port** is required, nor allowed; note that directory separators must be URL-encoded, like any other characters that are special to URLs; so the socket

/usr/local/var/ldapi

must be specified as

ldapi://%2Fusr%2Flocal%2Fvar%2Fldapi

A space separated list of URIs may be provided.

BASE <base>
Specifies the default base DN to use when performing ldap operations. The base must be specified as a Distinguished Name in LDAP format.

BINDDN <dn>
Specifies the default bind DN to use when performing ldap operations. The bind DN must be specified as a Distinguished Name in LDAP format. **This is a user−only option.**

DEREF <when>
Specifies how alias dereferencing is done when performing a search. The **<when>** can be specified as one of the following keywords:

never Aliases are never dereferenced. This is the default.

searching
Aliases are dereferenced in subordinates of the base object, but not in locating the base object of the search.

finding Aliases are only dereferenced when locating the base object of the search.

always Aliases are dereferenced both in searching and in locating the base object of the search.

HOST <name[:port] ...>
Specifies the name(s) of an LDAP server(s) to which the *LDAP* library should connect. Each server's name can be specified as a domain-style name or an IP address and optionally followed by a ':' and the port number the ldap server is listening on. A space separated list of hosts may be provided. **HOST** is deprecated in favor of **URI**.

NETWORK_TIMEOUT <integer>
Specifies the timeout (in seconds) after which the poll(2)/select(2) following a connect(2) returns in case of no activity.

PORT <port>
Specifies the default port used when connecting to LDAP servers(s). The port may be specified as a number. **PORT** is deprecated in favor of **URI.**

REFERRALS <on/true/yes/off/false/no>
Specifies if the client should automatically follow referrals returned by LDAP servers. The default is on. Note that the command line tools **ldapsearch**(1) &co always override this option.

SIZELIMIT <integer>
Specifies a size limit to use when performing searches. The number should be a non-negative integer. *SIZELIMIT* of zero (0) specifies unlimited search size.

TIMELIMIT <integer>
Specifies a time limit to use when performing searches. The number should be a non-negative integer. *TIMELIMIT* of zero (0) specifies unlimited search time to be used. **VERSION {2|3}** Specifies what version of the LDAP protocol should be used.

TIMEOUT <integer>
Specifies a timeout (in seconds) after which calls to synchronous LDAP APIs will abort if no response is received. Also used for any **ldap_result**(3) calls where a NULL timeout parameter is supplied.

SASL OPTIONS

If OpenLDAP is built with Simple Authentication and Security Layer support, there are more options you can specify.

SASL_MECH <mechanism>
Specifies the SASL mechanism to use. **This is a user−only option.**

SASL_REALM <realm>
Specifies the SASL realm. **This is a user−only option.**

SASL_AUTHCID <authcid>
Specifies the authentication identity. **This is a user−only option.**

SASL_AUTHZID <authcid>
Specifies the proxy authorization identity. **This is a user−only option.**

SASL_SECPROPS <properties>
Specifies Cyrus SASL security properties. The **<properties>** can be specified as a comma-separated list of the following:

none (without any other properties) causes the properties defaults ("noanonymous,noplain") to be cleared.

noplain
disables mechanisms susceptible to simple passive attacks.

noactive
disables mechanisms susceptible to active attacks.

nodict disables mechanisms susceptible to passive dictionary attacks.

noanonymous
disables mechanisms which support anonymous login.

forwardsec
requires forward secrecy between sessions.

passcred
requires mechanisms which pass client credentials (and allows mechanisms which can pass credentials to do so).

minssf=<factor>
specifies the minimum acceptable *security strength factor* as an integer approximating the effective key length used for encryption. 0 (zero) implies no protection, 1 implies integrity protection only, 56 allows DES or other weak ciphers, 112 allows triple DES and other strong ciphers, 128 allows RC4, Blowfish and other modern strong ciphers. The default is 0.

maxssf=<factor>
specifies the maximum acceptable *security strength factor* as an integer (see **minssf** description). The default is **INT_MAX**.

maxbufsize=<factor>
specifies the maximum security layer receive buffer size allowed. 0 disables security layers. The default is 65536.

TLS OPTIONS

If OpenLDAP is built with Transport Layer Security support, there are more options you can specify. These options are used when an **ldaps:// URI** is selected (by default or otherwise) or when the application negotiates TLS by issuing the LDAP StartTLS operation.

TLS_CACERT <filename>
Specifies the file that contains certificates for all of the Certificate Authorities the client will recognize.

TLS_CACERTDIR <path>
Specifies the path of a directory that contains Certificate Authority certificates in separate individual files. The **TLS_CACERT** is always used before **TLS_CACERTDIR.** This parameter is ignored with GNUtls.

TLS_CERT <filename>
Specifies the file that contains the client certificate. **This is a user−only option.**

TLS_KEY <filename>
Specifies the file that contains the private key that matches the certificate stored in the **TLS_CERT** file. Currently, the private key must not be protected with a password, so it is of critical importance that the key file is protected carefully. **This is a user−only option.**

TLS_CIPHER_SUITE <cipher-suite-spec>
Specifies acceptable cipher suite and preference order. <cipher-suite-spec> should be a cipher specification for OpenSSL, e.g., HIGH:MEDIUM:+SSLv2.

TLS_RANDFILE <filename>
Specifies the file to obtain random bits from when /dev/[u]random is not available. Generally set to the name of the EGD/PRNGD socket. The environment variable RANDFILE can also be used to specify the filename. This parameter is ignored with GNUtls.

TLS_REQCERT <level>
Specifies what checks to perform on server certificates in a TLS session, if any. The **<level>** can be specified as one of the following keywords:

never The client will not request or check any server certificate.

allow The server certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, it will be ignored and the session proceeds normally.

try The server certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, the session is immediately terminated.

demand | hard
These keywords are equivalent. The server certificate is requested. If no certificate is provided, or a bad certificate is provided, the session is immediately terminated. This is the default setting.

TLS_CRLCHECK <level>
Specifies if the Certificate Revocation List (CRL) of the CA should be used to verify if the server certificates have not been revoked. This requires **TLS_CACERTDIR** parameter to be set. This parameter is ignored with GNUtls. **<level>** can be specified as one of the following keywords:

none No CRL checks are performed

peer Check the CRL of the peer certificate

all Check the CRL for a whole certificate chain

TLS_CRLFILE <filename>
Specifies the file containing a Certificate Revocation List to be used to verify if the server certificates have not been revoked. This parameter is only supported with GNUtls.

ENVIRONMENT VARIABLES

LDAPNOINIT
disable all defaulting

LDAPCONF
: path of a configuration file

LDAPRC
: basename of ldaprc file in $HOME or $CWD

LDAP<option-name>
: Set <option-name> as from ldap.conf

FILES

/etc/openldap/ldap.conf
: system-wide ldap configuration file

$HOME/ldaprc, $HOME/.ldaprc
: user ldap configuration file

$CWD/ldaprc
: local ldap configuration file

SEE ALSO

ldap(3), **ldap_set_option**(3), **ldap_result**(3), **openssl**(1), **sasl**(3)

AUTHOR

Kurt Zeilenga, The OpenLDAP Project

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

ldap_table − Postfix LDAP client configuration

SYNOPSIS

postmap -q "*string***" ldap:/etc/postfix/filename**

postmap -q - ldap:/etc/postfix/*filename* <*inputfile*

DESCRIPTION

The Postfix mail system uses optional tables for address rewriting or mail routing. These tables are usually in **dbm** or **db** format.

Alternatively, lookup tables can be specified as LDAP databases.

In order to use LDAP lookups, define an LDAP source as a lookup table in main.cf, for example:

alias_maps = ldap:/etc/postfix/ldap-aliases.cf

The file /etc/postfix/ldap-aliases.cf has the same format as the Postfix main.cf file, and can specify the parameters described below. An example is given at the end of this manual.

This configuration method is available with Postfix version 2.1 and later. See the section "BACKWARDS COMPATIBILITY" below for older Postfix versions.

For details about LDAP SSL and STARTTLS, see the section on SSL and STARTTLS below.

BACKWARDS COMPATIBILITY

For backwards compatibility with Postfix version 2.0 and earlier, LDAP parameters can also be defined in main.cf. Specify as LDAP source a name that doesn't begin with a slash or a dot. The LDAP parameters will then be accessible as the name you've given the source in its definition, an underscore, and the name of the parameter. For example, if the map is specified as "ldap:*ldapsource*", the "server_host" parameter below would be defined in main.cf as "*ldapsource*_server_host".

Note: with this form, the passwords for the LDAP sources are written in main.cf, which is normally world-readable. Support for this form will be removed in a future Postfix version.

Postfix 2.2 has enhanced query interfaces for MySQL and PostgreSQL. These include features that were previously available only in the Postfix LDAP client. This work also created an opportunity for improvements in the LDAP interface. The primary compatibility issue is that **result_filter** (a name that has caused some confusion as to its meaning in the past) has been renamed to **result_format**. For backwards compatibility with the pre 2.2 LDAP client, **result_filter** can for now be used instead of **result_format**, when the latter parameter is not also set. The new name better reflects the function of the parameter. This compatibility interface may be removed in a future release.

LIST MEMBERSHIP

When using LDAP to store lists such as $mynetworks, $mydestination, $relay_domains, $local_recipient_maps, etc., it is important to understand that the table must store each list member as a separate key. The table lookup verifies the *existence* of the key. See "Postfix lists versus tables" in the DATABASE_README document for a discussion.

Do NOT create tables that return the full list of domains in $mydestination or $relay_domains etc., or IP addresses in $mynetworks.

DO create tables with each matching item as a key and with an arbitrary value. With LDAP databases it is not uncommon to return the key itself.

For example, NEVER do this in a map defining $mydestination:

```
query_filter = domain=*
result_attribute = domain
```

Do this instead:

```
query_filter = domain=%s
result_attribute = domain
```

GENERAL LDAP PARAMETERS

In the text below, default values are given in parentheses. Note: don't use quotes in these variables; at least, not until the Postfix configuration routines understand how to deal with quoted strings.

server_host (default: localhost)

The name of the host running the LDAP server, e.g.

```
server_host = ldap.example.com
```

Depending on the LDAP client library you're using, it should be possible to specify multiple servers here, with the library trying them in order should the first one fail. It should also be possible to give each server in the list a different port (overriding **server_port** below), by naming them like

```
server_host = ldap.example.com:1444
```

With OpenLDAP, a (list of) LDAP URLs can be used to specify both the hostname(s) and the port(s):

```
server_host = ldap://ldap.example.com:1444
        ldap://ldap2.example.com:1444
```

All LDAP URLs accepted by the OpenLDAP library are supported, including connections over UNIX domain sockets, and LDAP SSL (the last one provided that OpenLDAP was compiled with support for SSL):

```
server_host = ldapi://%2Fsome%2Fpath
        ldaps://ldap.example.com:636
```

server_port (default: 389)

The port the LDAP server listens on, e.g.

```
server_port = 778
```

timeout (default: 10 seconds)

The number of seconds a search can take before timing out, e.g.

```
timeout = 5
```

search_base (No default; you must configure this)

The RFC2253 base DN at which to conduct the search, e.g.

```
search_base = dc=your, dc=com
```

With Postfix 2.2 and later this parameter supports the following '%' expansions:

%% This is replaced by a literal '%' character.

%s This is replaced by the input key. RFC 2253 quoting is used to make sure that the input key does not add unexpected metacharacters.

%u When the input key is an address of the form user@domain, **%u** is replaced by the (RFC 2253) quoted local part of the address. Otherwise, **%u** is replaced by the entire search string. If the localpart is empty, the search is suppressed and returns no results.

%d When the input key is an address of the form user@domain, **%d** is replaced by the (RFC 2253) quoted domain part of the address. Otherwise, the search is suppressed and returns no results.

%[SUD]
For the **search_base** parameter, the upper-case equivalents of the above expansions behave identically to their lower-case counter-parts. With the **result_format** parameter (previously called **result_filter** see the COMPATIBILITY section and below), they expand to the corresponding components of input key rather than the result value.

%[1-9] The patterns %1, %2, ... %9 are replaced by the corresponding most significant component of the input key's domain. If the input key is *user@mail.example.com*, then %1 is **com**, %2 is **example** and %3 is **mail**. If the input key is unqualified or does not have enough domain components to satisfy all the specified patterns, the search is suppressed and returns no results.

query_filter (default: mailacceptinggeneralid=%s)
The RFC2254 filter used to search the directory, where **%s** is a substitute for the address Postfix is trying to resolve, e.g.

query_filter = (&(mail=%s)(paid_up=true))

This parameter supports the following '%' expansions:

%% This is replaced by a literal '%' character. (Postfix 2.2 and later).

%s This is replaced by the input key. RFC 2254 quoting is used to make sure that the input key does not add unexpected metacharacters.

%u When the input key is an address of the form user@domain, **%u** is replaced by the (RFC 2254) quoted local part of the address. Otherwise, **%u** is replaced by the entire search string. If the localpart is empty, the search is suppressed and returns no results.

%d When the input key is an address of the form user@domain, **%d** is replaced by the (RFC 2254) quoted domain part of the address. Otherwise, the search is suppressed and returns no results.

%[SUD]
The upper-case equivalents of the above expansions behave in the **query_filter** parameter identically to their lower-case counter-parts. With the **result_format** parameter (previously called **result_filter** see the COMPATIBILITY section and below), they expand to the corresponding components of input key rather than the result value.

The above %S, %U and %D expansions are available with Postfix 2.2 and later.

%[1-9] The patterns %1, %2, ... %9 are replaced by the corresponding most significant component of the input key's domain. If the input key is *user@mail.example.com*, then %1 is **com**, %2 is **example** and %3 is **mail**. If the input key is unqualified or does not have enough domain components to satisfy all the specified patterns, the search is suppressed and returns no results.

The above %1, ..., %9 expansions are available with Postfix 2.2 and later.

The "domain" parameter described below limits the input keys to addresses in matching domains. When the "domain" parameter is non-empty, LDAP queries for unqualified addresses or addresses in non-matching domains are suppressed and return no results.

NOTE: DO NOT put quotes around the **query_filter** parameter.

result_format (default: %s)

Called **result_filter** in Postfix releases prior to 2.2. Format template applied to result attributes. Most commonly used to append (or prepend) text to the result. This parameter supports the following '%' expansions:

%% This is replaced by a literal '%' character. (Postfix 2.2 and later).

%s This is replaced by the value of the result attribute. When result is empty it is skipped.

%u When the result attribute value is an address of the form user@domain, **%u** is replaced by the local part of the address. When the result has an empty localpart it is skipped.

%d When a result attribute value is an address of the form user@domain, **%d** is replaced by the domain part of the attribute value. When the result is unqualified it is skipped.

%[SUD1-9]

The upper-case and decimal digit expansions interpolate the parts of the input key rather than the result. Their behavior is identical to that described with **query_filter**, and in fact because the input key is known in advance, lookups whose key does not contain all the information specified in the result template are suppressed and return no results.

The above %S, %U, %D and %1, ..., %9 expansions are available with Postfix 2.2 and later.

For example, using "result_format = smtp:[%s]" allows one to use a mailHost attribute as the basis of a transport(5) table. After applying the result format, multiple values are concatenated as comma separated strings. The expansion_limit and size_limit parameters explained below allow one to restrict the number of values in the result, which is especially useful for maps that should return a single value.

The default value **%s** specifies that each attribute value should be used as is.

This parameter was called **result_filter** in Postfix releases prior to 2.2. If no "result_format" is specified, the value of "result_filter" will be used instead before resorting to the default value. This provides compatibility with old configuration files.

NOTE: DO NOT put quotes around the result format!

domain (default: no domain list)

This is a list of domain names, paths to files, or dictionaries. When specified, only fully qualified search keys with a *non-empty* localpart and a matching domain are eligible for lookup: 'user' lookups, bare domain lookups and "@domain" lookups are not performed. This can significantly reduce the query load on the LDAP server.

```
domain = postfix.org, hash:/etc/postfix/searchdomains
```

It is best not to use LDAP to store the domains eligible for LDAP lookups.

NOTE: DO NOT define this parameter for local(8) aliases.

This feature is available in Postfix 1.0 and later.

result_attribute (default: maildrop)

The attribute(s) Postfix will read from any directory entries returned by the lookup, to be resolved to an email address.

```
result_attribute = mailbox, maildrop
```

special_result_attribute (default: empty)

The attribute(s) of directory entries that can contain DNs or URLs. If found, a recursive subsequent search is done using their values.

```
special_result_attribute = memberdn
```

DN recursion retrieves the same result_attributes as the main query, including the special attributes for further recursion. URI processing retrieves only those attributes that are included in the URI definition and are *also* listed in "result_attribute". If the URI lists any of the map's special result attributes, these are also retrieved and used recursively.

terminal_result_attribute (default: empty)

When one or more terminal result attributes are found in an LDAP entry, all other result attributes are ignored and only the terminal result attributes are returned. This is useful for delegating expansion of group members to a particular host, by using an optional "maildrop" attribute on selected groups to route the group to a specific host, where the group is expanded, possibly via mailing-list manager or other special processing.

```
terminal_result_attribute = maildrop
```

This feature is available with Postfix 2.4 or later.

leaf_result_attribute (default: empty)

When one or more special result attributes are found in a non-terminal (see above) LDAP entry, leaf result attributes are excluded from the expansion of that entry. This is useful when expanding groups and the desired mail address attribute(s) of the member objects obtained via DN or URI recursion are also present in the group object. To only return the attribute values from the leaf objects and not the containing group, add the attribute to the leaf_result_attribute list, and not the result_attribute list, which is always expanded. Note, the default value of "result_attribute" is not empty, you may want to set it explicitly empty when using "leaf_result_attribute" to expand the group to a list of member DN addresses. If groups have both member DN references AND attributes that hold multiple string valued rfc822 addresses, then the string attributes go in "result_attribute". The attributes that represent the email addresses of objects referenced via a DN (or LDAP URI) go in "leaf_result_attribute".

```
result_attribute = memberaddr
special_result_attribute = memberdn
terminal_result_attribute = maildrop
leaf_result_attribute = mail
```

This feature is available with Postfix 2.4 or later.

scope (default: sub)

The LDAP search scope: **sub**, **base**, or **one**. These translate into LDAP_SCOPE_SUBTREE, LDAP_SCOPE_BASE, and LDAP_SCOPE_ONELEVEL.

bind (default: yes)

Whether or not to bind to the LDAP server. Newer LDAP implementations don't require clients to bind, which saves time. Example:

```
bind = no
```

If you do need to bind, you might consider configuring Postfix to connect to the local machine on a port that's an SSL tunnel to your LDAP server. If your LDAP server doesn't natively support SSL, put a tunnel (wrapper, proxy, whatever you want to call it) on that system too. This should prevent the password from traversing the network in the clear.

bind_dn (default: empty)
If you do have to bind, do it with this distinguished name. Example:

bind_dn = uid=postfix, dc=your, dc=com

bind_pw (default: empty)
The password for the distinguished name above. If you have to use this, you probably want to make the map configuration file readable only by the Postfix user. When using the obsolete ldap:ldapsource syntax, with map parameters in main.cf, it is not possible to securely store the bind password. This is because main.cf needs to be world readable to allow local accounts to submit mail via the sendmail command. Example:

bind_pw = postfixpw

cache (IGNORED with a warning)

cache_expiry (IGNORED with a warning)

cache_size (IGNORED with a warning)
The above parameters are NO LONGER SUPPORTED by Postfix. Cache support has been dropped from OpenLDAP as of release 2.1.13.

recursion_limit (default: 1000)
A limit on the nesting depth of DN and URL special result attribute evaluation. The limit must be a non-zero positive number.

expansion_limit (default: 0)
A limit on the total number of result elements returned (as a comma separated list) by a lookup against the map. A setting of zero disables the limit. Lookups fail with a temporary error if the limit is exceeded. Setting the limit to 1 ensures that lookups do not return multiple values.

size_limit (default: $expansion_limit)
A limit on the number of LDAP entries returned by any single LDAP search performed as part of the lookup. A setting of 0 disables the limit. Expansion of DN and URL references involves nested LDAP queries, each of which is separately subjected to this limit.

Note: even a single LDAP entry can generate multiple lookup results, via multiple result attributes and/or multi-valued result attributes. This limit caps the per search resource utilization on the LDAP server, not the final multiplicity of the lookup result. It is analogous to the "-z" option of "ldapsearch".

dereference (default: 0)
When to dereference LDAP aliases. (Note that this has nothing do with Postfix aliases.) The permitted values are those legal for the OpenLDAP/UM LDAP implementations:

0 never

1 when searching

2 when locating the base object for the search

3 always

See ldap.h or the ldap_open(3) or ldapsearch(1) man pages for more information. And if you're using an LDAP package that has other possible values, please bring it to the attention of the postfix-users@postfix.org mailing list.

chase_referrals (default: 0)
Sets (or clears) LDAP_OPT_REFERRALS (requires LDAP version 3 support).

version (default: 2)
Specifies the LDAP protocol version to use.

debuglevel (default: 0)
What level to set for debugging in the OpenLDAP libraries.

LDAP SSL AND STARTTLS PARAMETERS

If you're using the OpenLDAP libraries compiled with SSL support, Postfix can connect to LDAP SSL servers and can issue the STARTTLS command.

LDAP SSL service can be requested by using a LDAP SSL URL in the server_host parameter:

server_host = ldaps://ldap.example.com:636

STARTTLS can be turned on with the start_tls parameter:

start_tls = yes

Both forms require LDAP protocol version 3, which has to be set explicitly with:

version = 3

If any of the Postfix programs querying the map is configured in master.cf to run chrooted, all the certificates and keys involved have to be copied to the chroot jail. Of course, the private keys should only be readable by the user "postfix".

The following parameters are relevant to LDAP SSL and STARTTLS:

start_tls (default: no)
Whether or not to issue STARTTLS upon connection to the server. Don't set this with LDAP SSL (the SSL session is setup automatically when the TCP connection is opened).

tls_ca_cert_dir (No default; set either this or tls_ca_cert_file)
Directory containing X509 Certificate Authority certificates in PEM format which are to be recognized by the client in SSL/TLS connections. The files each contain one CA certificate. The files are looked up by the CA subject name hash value, which must hence be available. If more than one CA certificate with the same name hash value exist, the extension must be different (e.g. 9d66eef0.0, 9d66eef0.1 etc). The search is performed in the ordering of the extension number, regardless of other properties of the certificates. Use the c_rehash utility (from the OpenSSL distribution) to create the necessary links.

tls_ca_cert_file (No default; set either this or tls_ca_cert_dir)
File containing the X509 Certificate Authority certificates in PEM format which are to be recognized by the client in SSL/TLS connections. This setting takes precedence over tls_ca_cert_dir.

tls_cert (No default; you must set this)
File containing client's X509 certificate to be used by the client in SSL/ TLS connections.

tls_key (No default; you must set this)
File containing the private key corresponding to the above tls_cert.

tls_require_cert (default: no)
Whether or not to request server's X509 certificate and check its validity when establishing SSL/TLS connections.

tls_random_file (No default)
Path of a file to obtain random bits from when /dev/[u]random is not available, to be used by the client in SSL/TLS connections.

tls_cipher_suite (No default)
Cipher suite to use in SSL/TLS negotiations.

EXAMPLE

Here’s a basic example for using LDAP to look up local(8) aliases. Assume that in main.cf, you have:

```
alias_maps = hash:/etc/aliases,
    ldap:/etc/postfix/ldap-aliases.cf
```

and in ldap:/etc/postfix/ldap-aliases.cf you have:

```
server_host = ldap.example.com
search_base = dc=example, dc=com
```

Upon receiving mail for a local address "ldapuser" that isn’t found in the /etc/aliases database, Postfix will search the LDAP server listening at port 389 on ldap.example.com. It will bind anonymously, search for any directory entries whose mailacceptinggeneralid attribute is "ldapuser", read the "maildrop" attributes of those found, and build a list of their maildrops, which will be treated as RFC822 addresses to which the message will be delivered.

SEE ALSO

postmap(1), Postfix lookup table manager
postconf(5), configuration parameters
mysql_table(5), MySQL lookup tables
pgsql_table(5), PostgreSQL lookup tables

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview
LDAP_README, Postfix LDAP client guide

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Carsten Hoeger, Hery Rakotoarisoa, John Hensley, Keith Stevenson, LaMont Jones, Liviu Daia, Manuel Guesdon, Mike Mattice, Prabhat K Singh, Sami Haahtinen, Samuel Tardieu, Victor Duchovni, and many others.

NAME

ldif − LDAP Data Interchange Format

DESCRIPTION

The LDAP Data Interchange Format (LDIF) is used to represent LDAP entries and change records in text form. LDAP tools, such as **ldapadd**(1) and **ldapsearch**(1), read and write LDIF entry records. **ldapmodify**(1) reads LDIF change records.

This manual page provides a basic description of LDIF. A formal specification of LDIF is published in RFC 2849.

ENTRY RECORDS

LDIF entry records are used to represent directory entries. The basic form of an entry record is:

```
dn: <distinguished name>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
<attrdesc>:: <base64-encoded-value>
<attrdesc>:< <URL>
...
```

The value may be specified as UTF-8 text or as base64 encoded data, or a URI may be provided to the location of the attribute value.

A line may be continued by starting the next line with a single space or tab, e.g.,

```
dn: cn=Barbara J Jensen,dc=exam
 ple,dc=com
```

Lines beginning with a sharp sign (’#’) are ignored.

Multiple attribute values are specified on separate lines, e.g.,

```
cn: Barbara J Jensen
cn: Babs Jensen
```

If an value contains a non-printing character, or begins with a space or a colon ’:’, the <attrtype> is followed by a double colon and the value is encoded in base 64 notation. e.g., the value " begins with a space" would be encoded like this:

```
cn:: IGJlZ2lucyB3aXRoIGEgc3BhY2U=
```

If the attribute value is located in a file, the <attrtype> is followed by a ’:<’ and a file: URI. e.g., the value contained in the file /tmp/value would be listed like this:

```
cn:< file:///tmp/value
```

Other URI schemes (ftp,http) may be supported as well.

Multiple entries within the same LDIF file are separated by blank lines.

ENTRY RECORD EXAMPLE

Here is an example of an LDIF file containing three entries.

```
dn: cn=Barbara J Jensen,dc=example,dc=com
cn: Barbara J Jensen
cn: Babs Jensen
objectclass: person
description:< file:///tmp/babs
sn: Jensen

dn: cn=Bjorn J Jensen,dc=example,dc=com
cn: Bjorn J Jensen
cn: Bjorn Jensen
objectclass: person
sn: Jensen
```

```
dn: cn=Jennifer J Jensen,dc=example,dc=com
cn: Jennifer J Jensen
cn: Jennifer Jensen
objectclass: person
sn: Jensen
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALD
 A4MChAODQ4SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQ
 ERXRTc4UG1RV19iZ2hnPk1xeXBkeFxlZ2P/2wBDARESEhgVG
...
```

Note that the description in Barbara Jensen's entry is read from file:///tmp/babs and the jpegPhoto in Jennifer Jensen's entry is encoded using base 64.

CHANGE RECORDS

LDIF change records are used to represent directory change requests. Each change record starts with line indicating the distinguished name of the entry being changed:

```
dn: <distinguishedname>
```

```
changetype: <[modify|add|delete|modrdn]>
```

Finally, the change information itself is given, the format of which depends on what kind of change was specified above. For a *changetype* of *modify*, the format is one or more of the following:

```
add: <attributetype>
<attrdesc>: <value1>
<attrdesc>: <value2>
...
-
```

Or, for a replace modification:

```
replace: <attributetype>
<attrdesc>: <value1>
<attrdesc>: <value2>
...
-
```

If no *attributetype* lines are given to replace, the entire attribute is to be deleted (if present).

Or, for a delete modification:

```
delete: <attributetype>
<attrdesc>: <value1>
<attrdesc>: <value2>
...
-
```

If no *attributetype* lines are given to delete, the entire attribute is to be deleted.

For a *changetype* of *add*, the format is:

```
<attrdesc1>: <value1>
<attrdesc1>: <value2>
...
<attrdescN>: <value1>
<attrdescN>: <value2>
```

For a *changetype* of *modrdn* or *moddn*, the format is:

```
newrdn: <newrdn>
deleteoldrdn: 0 | 1
newsuperior: <DN>
```

where a value of 1 for deleteoldrdn means to delete the values forming the old rdn from the entry, and a

value of 0 means to leave the values as non-distinguished attributes in the entry. The newsuperior line is optional and, if present, specifies the new superior to move the entry to.

For a *changetype* of *delete*, no additional information is needed in the record.

Note that attribute values may be presented using base64 or in files as described for entry records. Lines in change records may be continued in the manner described for entry records as well.

CHANGE RECORD EXAMPLE

The following sample LDIF file contains a change record of each type of change.

```
dn: cn=Babs Jensen,dc=example,dc=com
changetype: add
objectclass: person
objectclass: extensibleObject
cn: babs
cn: babs jensen
sn: jensen

dn: cn=Babs Jensen,dc=example,dc=com
changetype: modify
add: givenName
givenName: Barbara
givenName: babs
-
replace: description
description: the fabulous babs
-
delete: sn
sn: jensen
-

dn: cn=Babs Jensen,dc=example,dc=com
changetype: modrdn
newrdn: cn=Barbara J Jensen
deleteoldrdn: 0
newsuperior: ou=People,dc=example,dc=com

dn: cn=Barbara J Jensen,ou=People,dc=example,dc=com
changetype: delete
```

INCLUDE STATEMENT

The LDIF parser has been extended to support an **include** statement for referencing other LDIF files. The **include** statement must be separated from other records by a blank line. The referenced file is specified using a file: URI and all of its contents are incorporated as if they were part of the original LDIF file. As above, other URI schemes may be supported. For example:

```
dn: dc=example,dc=com
objectclass: domain
dc: example

include: file:///tmp/example.com.ldif

dn: dc=example,dc=org
objectclass: domain
dc: example
```

This feature is not part of the LDIF specification in RFC 2849 but is expected to appear in a future revision

of this spec. It is supported by the **ldapadd**(1), **ldapmodify**(1), and **slapadd**(8) commands.

SEE ALSO

ldap(3), **ldapsearch**(1), **ldapadd**(1), **ldapmodify**(1), **slapadd**(8), **slapcat**(8), **slapd-ldif**(5), **slapd.replog**(5).

"LDAP Data Interchange Format," Good, G., RFC 2849.

ACKNOWLEDGEMENTS

NAME

ldif − LDAP Data Interchange Format

DESCRIPTION

The LDAP Data Interchange Format (LDIF) is used to represent LDAP entries and change records in text form. LDAP tools, such as **ldapadd**(1) and **ldapsearch**(1), read and write LDIF entry records. **ldapmodify**(1) reads LDIF change records.

This manual page provides a basic description of LDIF. A formal specification of LDIF is published in RFC 2849.

ENTRY RECORDS

LDIF entry records are used to represent directory entries. The basic form of an entry record is:

```
dn: <distinguished name>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>
<attrdesc>:: <base64-encoded-value>
<attrdesc>:< <URL>
...
```

The value may be specified as UTF-8 text or as base64 encoded data, or a URI may be provided to the location of the attribute value.

A line may be continued by starting the next line with a single space or tab, e.g.,

```
dn: cn=Barbara J Jensen,dc=exam
 ple,dc=com
```

Lines beginning with a sharp sign ('#') are ignored.

Multiple attribute values are specified on separate lines, e.g.,

```
cn: Barbara J Jensen
cn: Babs Jensen
```

If an value contains a non-printing character, or begins with a space or a colon ':', the <attrtype> is followed by a double colon and the value is encoded in base 64 notation. e.g., the value " begins with a space" would be encoded like this:

```
cn:: IGJlZ2lucyB3aXRoIGEgc3BhY2U=
```

If the attribute value is located in a file, the <attrtype> is followed by a ':<' and a file: URI. e.g., the value contained in the file /tmp/value would be listed like this:

```
cn:< file:///tmp/value
```

Other URI schemes (ftp,http) may be supported as well.

Multiple entries within the same LDIF file are separated by blank lines.

ENTRY RECORD EXAMPLE

Here is an example of an LDIF file containing three entries.

```
dn: cn=Barbara J Jensen,dc=example,dc=com
cn: Barbara J Jensen
cn: Babs Jensen
objectclass: person
description:< file:///tmp/babs
sn: Jensen

dn: cn=Bjorn J Jensen,dc=example,dc=com
cn: Bjorn J Jensen
cn: Bjorn Jensen
objectclass: person
sn: Jensen
```

```
dn: cn=Jennifer J Jensen,dc=example,dc=com
cn: Jennifer J Jensen
cn: Jennifer Jensen
objectclass: person
sn: Jensen
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALD
 A4MChAODQ4SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQ
 ERXRTc4UG1RV19iZ2hnPk1xeXBkeFxlZ2P/2wBDARESEhgVG
...
```

Note that the description in Barbara Jensen's entry is read from file:///tmp/babs and the jpegPhoto in Jennifer Jensen's entry is encoded using base 64.

CHANGE RECORDS

LDIF change records are used to represent directory change requests. Each change record starts with line indicating the distinguished name of the entry being changed:

```
dn: <distinguishedname>
```

```
changetype: <[modify|add|delete|modrdn]>
```

Finally, the change information itself is given, the format of which depends on what kind of change was specified above. For a *changetype* of *modify*, the format is one or more of the following:

```
add: <attributetype>
<attrdesc>: <value1>
<attrdesc>: <value2>
...
-
```

Or, for a replace modification:

```
replace: <attributetype>
<attrdesc>: <value1>
<attrdesc>: <value2>
...
-
```

If no *attributetype* lines are given to replace, the entire attribute is to be deleted (if present).

Or, for a delete modification:

```
delete: <attributetype>
<attrdesc>: <value1>
<attrdesc>: <value2>
...
-
```

If no *attributetype* lines are given to delete, the entire attribute is to be deleted.

For a *changetype* of *add*, the format is:

```
<attrdesc1>: <value1>
<attrdesc1>: <value2>
...
<attrdescN>: <value1>
<attrdescN>: <value2>
```

For a *changetype* of *modrdn* or *moddn*, the format is:

```
newrdn: <newrdn>
deleteoldrdn: 0 | 1
newsuperior: <DN>
```

where a value of 1 for deleteoldrdn means to delete the values forming the old rdn from the entry, and a

value of 0 means to leave the values as non-distinguished attributes in the entry. The newsuperior line is optional and, if present, specifies the new superior to move the entry to.

For a *changetype* of *delete*, no additional information is needed in the record.

Note that attribute values may be presented using base64 or in files as described for entry records. Lines in change records may be continued in the manner described for entry records as well.

CHANGE RECORD EXAMPLE

The following sample LDIF file contains a change record of each type of change.

```
dn: cn=Babs Jensen,dc=example,dc=com
changetype: add
objectclass: person
objectclass: extensibleObject
cn: babs
cn: babs jensen
sn: jensen

dn: cn=Babs Jensen,dc=example,dc=com
changetype: modify
add: givenName
givenName: Barbara
givenName: babs
-
replace: description
description: the fabulous babs
-
delete: sn
sn: jensen
-

dn: cn=Babs Jensen,dc=example,dc=com
changetype: modrdn
newrdn: cn=Barbara J Jensen
deleteoldrdn: 0
newsuperior: ou=People,dc=example,dc=com

dn: cn=Barbara J Jensen,ou=People,dc=example,dc=com
changetype: delete
```

INCLUDE STATEMENT

The LDIF parser has been extended to support an **include** statement for referencing other LDIF files. The **include** statement must be separated from other records by a blank line. The referenced file is specified using a file: URI and all of its contents are incorporated as if they were part of the original LDIF file. As above, other URI schemes may be supported. For example:

```
dn: dc=example,dc=com
objectclass: domain
dc: example

include: file:///tmp/example.com.ldif

dn: dc=example,dc=org
objectclass: domain
dc: example
```

This feature is not part of the LDIF specification in RFC 2849 but is expected to appear in a future revision

of this spec. It is supported by the **ldapadd**(1), **ldapmodify**(1), and **slapadd**(8) commands.

SEE ALSO

ldap(3), **ldapsearch**(1), **ldapadd**(1), **ldapmodify**(1), **slapadd**(8), **slapcat**(8), **slapd-ldif**(5), **slapd.replog**(5).

"LDAP Data Interchange Format," Good, G., RFC 2849.

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

libarchive-formats — archive formats supported by the libarchive library

DESCRIPTION

The libarchive(3) library reads and writes a variety of streaming archive formats. Generally speaking, all of these archive formats consist of a series of "entries". Each entry stores a single file system object, such as a file, directory, or symbolic link.

The following provides a brief description of each format supported by libarchive, with some information about recognized extensions or limitations of the current library support. Note that just because a format is supported by libarchive does not imply that a program that uses libarchive will support that format. Applications that use libarchive specify which formats they wish to support.

Tar Formats

The libarchive(3) library can read most tar archives. However, it only writes POSIX-standard "ustar" and "pax interchange" formats.

All tar formats store each entry in one or more 512-byte records. The first record is used for file metadata, including filename, timestamp, and mode information, and the file data is stored in subsequent records. Later variants have extended this by either appropriating undefined areas of the header record, extending the header to multiple records, or by storing special entries that modify the interpretation of subsequent entries.

gnutar The libarchive(3) library can read GNU-format tar archives. It currently supports the most popular GNU extensions, including modern long filename and linkname support, as well as atime and ctime data. The libarchive library does not support multi-volume archives, nor the old GNU long filename format. It can read GNU sparse file entries, including the new POSIX-based formats, but cannot write GNU sparse file entries.

pax The libarchive(3) library can read and write POSIX-compliant pax interchange format archives. Pax interchange format archives are an extension of the older ustar format that adds a separate entry with additional attributes stored as key/value pairs. The presence of this additional entry is the only difference between pax interchange format and the older ustar format. The extended attributes are of unlimited length and are stored as UTF-8 Unicode strings. Keywords defined in the standard are in all lowercase; vendors are allowed to define custom keys by preceding them with the vendor name in all uppercase. When writing pax archives, libarchive uses many of the SCHILY keys defined by Joerg Schilling's "star" archiver. The libarchive library can read most of the SCHILY keys. It silently ignores any keywords that it does not understand.

restricted pax
The libarchive library can also write pax archives in which it attempts to suppress the extended attributes entry whenever possible. The result will be identical to a ustar archive unless the extended attributes entry is required to store a long file name, long linkname, extended ACL, file flags, or if any of the standard ustar data (user name, group name, UID, GID, etc) cannot be fully represented in the ustar header. In all cases, the result can be dearchived by any program that can read POSIX-compliant pax interchange format archives. Programs that correctly read ustar format (see below) will also be able to read this format; any extended attributes will be extracted as separate files stored in PaxHeader directories.

ustar The libarchive library can both read and write this format. This format has the following limitations:

- Device major and minor numbers are limited to 21 bits. Nodes with larger numbers will not be added to the archive.

- Path names in the archive are limited to 255 bytes. (Shorter if there is no / character in exactly the right place.)
- Symbolic links and hard links are stored in the archive with the name of the referenced file. This name is limited to 100 bytes.
- Extended attributes, file flags, and other extended security information cannot be stored.
- Archive entries are limited to 2 gigabytes in size.

Note that the pax interchange format has none of these restrictions.

The libarchive library can also read a variety of commonly-used extensions to the basic tar format. In particular, it supports base-256 values in certain numeric fields. This essentially removes the limitations on file size, modification time, and device numbers.

The first tar program appeared in Seventh Edition Unix in 1979. The first official standard for the tar file format was the "ustar" (Unix Standard Tar) format defined by POSIX in 1988. POSIX.1-2001 extended the ustar format to create the "pax interchange" format.

Cpio Formats

The libarchive library can read a number of common cpio variants and can write "odc" and "newc" format archives. A cpio archive stores each entry as a fixed-size header followed by a variable-length filename and variable-length data. Unlike tar, cpio does only minimal padding of the header or file data. There are a variety of cpio formats, which differ primarily in how they store the initial header: some store the values as octal or hexadecimal numbers in ASCII, others as binary values of varying byte order and length.

binary The libarchive library can read both big-endian and little-endian variants of the original binary cpio format. This format used 32-bit binary values for file size and mtime, and 16-bit binary values for the other fields.

odc The libarchive library can both read and write this POSIX-standard format. This format stores the header contents as octal values in ASCII. It is standard, portable, and immune from byte-order confusion. File sizes and mtime are limited to 33 bits (8GB file size), other fields are limited to 18 bits.

SVR4 The libarchive library can read both CRC and non-CRC variants of this format. The SVR4 format uses eight-digit hexadecimal values for all header fields. This limits file size to 4GB, and also limits the mtime and other fields to 32 bits. The SVR4 format can optionally include a CRC of the file contents, although libarchive does not currently verify this CRC.

Cpio first appeared in PWB/UNIX 1.0, which was released within AT&T in 1977. PWB/UNIX 1.0 formed the basis of System III Unix, released outside of AT&T in 1981. This makes cpio older than tar, although cpio was not included in Version 7 AT&T Unix. As a result, the tar command became much better known in universities and research groups that used Version 7. The combination of the **find** and **cpio** utilities provided very precise control over file selection. Unfortunately, the format has many limitations that make it unsuitable for widespread use. Only the POSIX format permits files over 4GB, and its 18-bit limit for most other fields makes it unsuitable for modern systems. In addition, cpio formats only store numeric UID/GID values (not usernames and group names), which can make it very difficult to correctly transfer archives across systems with dissimilar user numbering.

Shar Formats

A "shell archive" is a shell script that, when executed on a POSIX-compliant system, will recreate a collection of file system objects. The libarchive library can write two different kinds of shar archives:

shar The traditional shar format uses a limited set of POSIX commands, including `echo`(1), `mkdir`(1), and `sed`(1). It is suitable for portably archiving small collections of plain text files. However, it is not generally well-suited for large archives (many implementations of `sh`(1) have limits on the size of a script) nor should it be used with non-text files.

shardump
: This format is similar to shar but encodes files using `uuencode`(1) so that the result will be a plain text file regardless of the file contents. It also includes additional shell commands that attempt to reproduce as many file attributes as possible, including owner, mode, and flags. The additional commands used to restore file attributes make shardump archives less portable than plain shar archives.

ISO9660 format
: Libarchive can read and extract from files containing ISO9660-compliant CDROM images. It also has partial support for Rockridge extensions. In many cases, this can remove the need to burn a physical CDROM. It also avoids security and complexity issues that come with virtual mounts and loopback devices.

Zip format
: Libarchive can extract from most zip format archives. It currently only supports uncompressed entries and entries compressed with the "deflate" algorithm. Older zip compression algorithms are not supported.

Archive (library) file format
: The Unix archive format (commonly created by the `ar`(1) archiver) is a general-purpose format which is used almost exclusively for object files to be read by the link editor `ld`(1). The ar format has never been standardised. There are two common variants: the GNU format derived from SVR4, and the BSD format, which first appeared in 4.4BSD. Libarchive provides read and write support for both variants.

mtree
: Libarchive can read files in `mtree`(5) format. This format is not a true archive format, but rather a description of a file hierarchy. When requested, libarchive obtains the contents of the files described by the `mtree`(5) format from files on disk instead.

SEE ALSO

`ar`(1), `cpio`(1), `mkisofs`(1), `shar`(1), `tar`(1), `zip`(1), `zlib`(3), `cpio`(5), `mtree`(5), `tar`(5)

NAME

link — dynamic loader and link editor interface

SYNOPSIS

```
#include <link.h>
```

DESCRIPTION

The include file ⟨link.h⟩ declares several structures that are present in dynamically linked programs and libraries. The structures define the interface between several components of the link-editor and loader mechanism. The layout of a number of these structures within the binaries resembles the a.out(5) format in many places as it serves such similar functions as symbol definitions (including the accompanying string table) and relocation records needed to resolve references to external entities.

It also records a number of data structures unique to the dynamic loading and linking process. These include references to other objects that are required to complete the link-editing process and indirection tables to facilitate *Position Independent Code* (PIC) to improve sharing of code pages among different processes.

The collection of data structures described here will be referred to as the *Run-time Relocation Section* (RRS) and is embedded in the standard text and data segments of the dynamically linked program or shared object image as the existing a.out(5) format offers no room for it elsewhere.

Several utilities cooperate to ensure that the task of getting a program ready to run can complete successfully in a way that optimizes the use of system resources. The compiler emits PIC code from which shared libraries can be built by ld(1). The compiler also includes size information of any initialized data items through the .size assembler directive.

PIC code differs from conventional code in that it accesses data variables through an indirection table, the Global Offset Table, by convention accessible by the reserved name *_GLOBAL_OFFSET_TABLE_*. The exact mechanism used for this is machine dependent, usually a machine register is reserved for the purpose. The rational behind this construct is to generate code that is independent of the actual load address. Only the values contained in the Global Offset Table may need updating at run-time depending on the load addresses of the various shared objects in the address space.

Likewise, procedure calls to globally defined functions are redirected through the Procedure Linkage Table (PLT) residing in the data segment of the core image. Again, this is done to avoid run-time modifications to the text segment.

The linker-editor allocates the Global Offset Table and Procedure Linkage Table when combining PIC object files into an image suitable for mapping into the process address space. It also collects all symbols that may be needed by the run-time link-editor and stores these along with the image's text and data bits. Another reserved symbol, *_DYNAMIC* is used to indicate the presence of the run-time linker structures. Whenever *_DYNAMIC* is relocated to 0, there is no need to invoke the run-time link-editor. If this symbol is non-zero, it points at a data structure from which the location of the necessary relocation- and symbol information can be derived. This is most notably used by the start-up module, *crt0*. The _DYNAMIC structure is conventionally located at the start of the data segment of the image to which it pertains.

DATA STRUCTURES

The data structures supporting dynamic linking and run-time relocation reside both in the text and data segments of the image they apply to. The text segments contain read-only data such as symbols descriptions and names, while the data segments contain the tables that need to be modified by during the relocation process.

The _DYNAMIC symbol references a *_dynamic* structure:

```
struct _dynamic {
        int     d_version;
        struct  so_debug *d_debug;
        union {
                struct section_dispatch_table *d_sdt;
        } d_un;
        struct  ld_entry *d_entry;
};
```

d_version This field provides for different versions of the dynamic linking implementation. The current version numbers understood by ld and ld.so are *LD_VERSION_SUN (3)*, which is used by the SunOS 4.x releases, and *LD_VERSION_BSD (8)*, which is currently in use by NetBSD.

d_un Refers to a *d_version* dependent data structure.

d_debug this field provides debuggers with a hook to access symbol tables of shared objects loaded as a result of the actions of the run-time link-editor.

d_entry this field is obsoleted by CRT interface version CRT_VERSION_BSD4, and is replaced by the crt_ldentry in *crt_ldso*.

The *section_dispatch_table* structure is the main "dispatcher" table, containing offsets into the image's segments where various symbol and relocation information is located.

```
struct section_dispatch_table {
        struct  so_map *sdt_loaded;
        long    sdt_sods;
        long    sdt_paths;
        long    sdt_got;
        long    sdt_plt;
        long    sdt_rel;
        long    sdt_hash;
        long    sdt_nzlist;
        long    sdt_filler2;
        long    sdt_buckets;
        long    sdt_strings;
        long    sdt_str_sz;
        long    sdt_text_sz;
        long    sdt_plt_sz;
};
```

sdt_loaded A pointer to the first link map loaded (see below). This field is set by ld.so(1) for the benefit of debuggers that may use it to load a shared object's symbol table.

sdt_sods The start of a (linked) list of shared object descriptors needed by *this* object.

sdt_paths Library search rules. A colon separated list of directories corresponding to the **-R** option of ld(1).

sdt_got The location of the Global Offset Table within this image.

sdt_plt The location of the Procedure Linkage Table within this image.

sdt_rel The location of an array of *relocation_info* structures (see a.out(5)) specifying run-time relocations.

sdt_hash The location of the hash table for fast symbol lookup in this object's symbol table.

sdt_nzlist The location of the symbol table.

sdt_filler2
Currently unused.

sdt_buckets
The number of buckets in *sdt_hash*

sdt_strings
The location of the symbol string table that goes with *sdt_nzlist*.

sdt_str_sz The size of the string table.

sdt_text_sz
The size of the object's text segment.

sdt_plt_sz The size of the Procedure Linkage Table.

A *sod* structure describes a shared object that is needed to complete the link edit process of the object containing it. A list of such objects (chained through *sod_next*) is pointed at by the *sdt_sods* in the section_dispatch_table structure.

```
struct sod {
        long    sod_name;
        u_int   sod_library : 1,
                sod_unused : 31;
        short   sod_major;
        short   sod_minor;
        long    sod_next;
};
```

sod_name The offset in the text segment of a string describing this link object.

sod_library If set, *sod_name* specifies a library that is to be searched for by ld.so. The path name is obtained by searching a set of directories (see also ldconfig(8)) for a shared object matching *lib<sod_name>.so.n.m*. If not set, *sod_name* should point at a full path name for the desired shared object.

sod_major Specifies the major version number of the shared object to load.

sod_minor Specifies the preferred minor version number of the shared object to load.

The run-time link-editor maintains a list of structures called *link maps* to keep track of all shared objects loaded into a process' address space. These structures are only used at run-time and do not occur within the text or data segment of an executable or shared library.

```
struct so_map {
        void    *som_addr;
        char    *som_path;
        struct  so_map *som_next;
        struct  sod *som_sod;
        void *som_sodbase;
        u_int   som_write : 1;
        struct  _dynamic *som_dynamic;
        void    *som_spd;
};
```

som_addr The address at which the shared object associated with this link map has been loaded.

som_path The full path name of the loaded object.

som_next Pointer to the next link map.

som_sod The *sod* structure that was responsible for loading this shared object.

som_sodbase Tossed in later versions the run-time linker.

som_write Set if (some portion of) this object's text segment is currently writable.

som_dynamic Pointer to this object's *_dynamic* structure.

som_spd Hook for attaching private data maintained by the run-time link-editor.

Symbol description with size. This is simply an *nlist* structure with one field (*nz_size*) added. Used to convey size information on items in the data segment of shared objects. An array of these lives in the shared object's text segment and is addressed by the *sdt_nzlist* field of *section_dispatch_table*.

```
struct nzlist {
        struct nlist   nlist;
        u_long         nz_size;
#define nz_un          nlist.n_un
#define nz_strx                nlist.n_un.n_strx
#define nz_name                nlist.n_un.n_name
#define nz_type                nlist.n_type
#define nz_value       nlist.n_value
#define nz_desc                nlist.n_desc
#define nz_other       nlist.n_other
};
```

nlist (see nlist(3)).

nz_size The size of the data represented by this symbol.

A hash table is included within the text segment of shared object to facilitate quick lookup of symbols during run-time link-editing. The *sdt_hash* field of the *section_dispatch_table* structure points at an array of *rrs_hash* structures:

```
struct rrs_hash {
        int     rh_symbolnum;          /* symbol number */
        int     rh_next;               /* next hash entry */
};
```

rh_symbolnum The index of the symbol in the shared object's symbol table (as given by the *ld_symbols* field).

rh_next In case of collisions, this field is the offset of the next entry in this hash table bucket. It is zero for the last bucket element.

The *rt_symbol* structure is used to keep track of run-time allocated commons and data items copied from shared objects. These items are kept on linked list and is exported through the *dd_cc* field in the *so_debug* structure (see below) for use by debuggers.

```
struct rt_symbol {
        struct nzlist          *rt_sp;
        struct rt_symbol       *rt_next;
        struct rt_symbol       *rt_link;
        void                   *rt_srcaddr;
```

```
        struct so_map          *rt_smp;
};
```

rt_sp The symbol description.

rt_next Virtual address of next rt_symbol.

rt_link Next in hash bucket. Used by internally by ld.so.

rt_srcaddr Location of the source of initialized data within a shared object.

rt_smp The shared object which is the original source of the data that this run-time symbol describes.

The *so_debug* structure is used by debuggers to gain knowledge of any shared objects that have been loaded in the process's address space as a result of run-time link-editing. Since the run-time link-editor runs as a part of process initialization, a debugger that wishes to access symbols from shared objects can only do so after the link-editor has been called from crt0. A dynamically linked binary contains a *so_debug* structure which can be located by means of the *d_debug* field in *_dynamic*.

```
struct  so_debug {
        int     dd_version;
        int     dd_in_debugger;
        int     dd_sym_loaded;
        char    *dd_bpt_addr;
        int     dd_bpt_shadow;
        struct rt_symbol *dd_cc;
};
```

dd_version Version number of this interface.

dd_in_debugger Set by the debugger to indicate to the run-time linker that the program is run under control of a debugger.

dd_sym_loaded Set by the run-time linker whenever it adds symbols by loading shared objects.

dd_bpt_addr The address were a breakpoint will be set by the run-time linker to divert control to the debugger. This address is determined by the start-up module, *crt0.o*, to be some convenient place before the call to _main.

dd_bpt_shadow Contains the original instruction that was at *dd_bpt_addr*. The debugger is expected to put this instruction back before continuing the program.

dd_cc A pointer to the linked list of run-time allocated symbols that the debugger may be interested in.

The *ld_entry* structure defines a set of service routines within ld.so. See dlfcn(3) for more information.

```
struct ld_entry {
        void    *(*dlopen)(char *, int);
        int     (*dlclose)(void *);
        void    *(*dlsym)(void *, char *);
        int     (*dlctl)(void *, int, void *);
        void    (*dlexit)(void);
};
```

The *crt_ldso* structure defines the interface between ld.so and the start-up code in crt0.

```
struct crt_ldso {
        int             crt_ba;
```

```
        int             crt_dzfd;
        int             crt_ldfd;
        struct _dynamic         *crt_dp;
        char            **crt_ep;
        void            *crt_bp;
        char            *crt_prog;
        char            *crt_ldso;
        char            *crt_ldentry;
};
#define CRT_VERSION_SUN                 1
#define CRT_VERSION_BSD2        2
#define CRT_VERSION_BSD3        3
#define CRT_VERSION_BSD4        4
```

crt_ba The virtual address at which ld.so was loaded by crt0.

crt_dzfd On SunOS systems, this field contains an open file descriptor to "/dev/zero" used to get demand paged zeroed pages. On NetBSD systems it contains -1.

crt_ldfd Contains an open file descriptor that was used by crt0 to load ld.so.

crt_dp A pointer to main's *_dynamic* structure.

crt_ep A pointer to the environment strings.

crt_bp The address at which a breakpoint will be placed by the run-time linker if the main program is run by a debugger. See *so_debug*

crt_prog The name of the main program as determined by crt0 (CRT_VERSION_BSD3 only).

crt_ldso The path of the run-time linker as mapped by crt0 (CRT_VERSION_BSD4 only).

crt_ldentry
The dlfcn(3) entry points provided by the run-time linker (CRT_VERSION_BSD4 only).

The *hints_header* and *hints_bucket* structures define the layout of the library hints, normally found in "/var/run/ld.so.hints", which is used by ld.so to quickly locate the shared object images in the file system. The organization of the hints file is not unlike that of an a.out(5) object file, in that it contains a header determining the offset and size of a table of fixed sized hash buckets and a common string pool.

```
struct hints_header {
        long            hh_magic;
#define HH_MAGIC        011421044151
        long            hh_version;
#define LD_HINTS_VERSION_1    1
#define LD_HINTS_VERSION_2    2
        long            hh_hashtab;
        long            hh_nbucket;
        long            hh_strtab;
        long            hh_strtab_sz;
        long            hh_ehints;
        long            hh_dirlist;
};
```

hh_magic Hints file magic number.

hh_version Interface version number.

hh_hashtab Offset of hash table.

hh_strtab Offset of string table.

hh_strtab_sz Size of strings.

hh_ehints Maximum usable offset in hints file.

hh_dirlist Offset in string table of a colon-separated list of directories that was used in constructing the hints file. See also `ldconfig`(8). This field is only available with interface version number `LD_HINTS_VERSION_2` and higher.

```
/*
 * Hash table element in hints file.
 */
struct hints_bucket {
        int             hi_namex;
        int             hi_pathx;
        int             hi_dewey[MAXDEWEY];
        int             hi_ndewey;
#define hi_major hi_dewey[0]
#define hi_minor hi_dewey[1]
        int             hi_next;
};
```

hi_namex Index of the string identifying the library.

hi_pathx Index of the string representing the full path name of the library.

hi_dewey The version numbers of the shared library.

hi_ndewey The number of valid entries in *hi_dewey*.

hi_next Next bucket in case of hashing collisions.

NAME

lkm.conf — loadable kernel module configuration file

DESCRIPTION

The **lkm.conf** file specifies loadable kernel modules, see lkm(4), that are to be loaded a boot time. The **lkm.conf** file is processed by /etc/rc.lkm at system boot time, if it exists.

Each line of the file is of the form

```
path options entry postinstall output when
```

Except for the *path*, all other fields can be "-" to indicate empty.

The *path* is either an absolute pathname, or the name of a file in /lkm or /usr/lkm that is the LKM to be loaded.

The *options* are some combination of the -d and -v options to modload(8).

The *entry* is the C symbol to call to initialize the module, defaulting to **xxxinit**().

The *postinstall* script is run after the LKM is installed.

The output of the ld(1) command is stored in *output*.

The *when* field specifies at which time in the startup process the LKM is loaded. There are three predefined values which correspond to three points in the processing of the system startup script /etc/rc:

BEFORENET
: Before networking is started. This works only if the /usr file system can already be mounted at this time.

BEFOREMOUNT
: Before all file systems listed in /etc/fstab are mounted (the /usr and /var file systems are already present).

AFTERMOUNT
: After all file systems are mounted.

A "-" entry defaults to BEFORENET.

FILES

/etc/lkm.conf, /etc/rc, /etc/rc.d/lkm, /etc/rc.lkm

SEE ALSO

lkm(4), modload(8)

HISTORY

The **lkm.conf** file appeared in NetBSD 1.3.

NAME

locale.alias — locale alias file

SYNOPSIS

locale.alias

DESCRIPTION

The **locale.alias** file describes locale aliases. Each line of this file can be described as the following BNF:

```
line                    :=      src ws dst
src                     :=      locale_name
                        |       locale_category_name
ws                      :=      <white spaces>
dst                     :=      locale_name
                        |       "/FORCE"
locale_category_name    :=      locale_name '/' category_name
category_name           :=      "LC_CTYPE"
                        |       "LC_COLLATE"
                        |       "LC_TIME"
                        |       "LC_NUMERIC"
                        |       "LC_MONETARY"
                        |       "LC_MESSAGES"
locale_name             :=      <locale name>
```

FILES

`/usr/share/locale/locale.alias` This file.

EXAMPLES

```
ja_JP.UTF-8/LC_CTYPE            en_US.UTF-8
```

This means that "`ja_JP.UTF-8`" for `LC_CTYPE` category is redirected to "`en_US.UTF-8`".

```
Pig                             /FORCE
```

This means that "`Pig`" for all categories is forcibly enabled.

SEE ALSO

`locale`(1), `nls`(7)

HISTORY

The **locale.alias** file appeared in NetBSD 3.0.

NAME

locate.conf — locate database configuration file

DESCRIPTION

The **locate.conf** file specifies the behavior of locate.updatedb(8), which creates the locate(1) database.

The **locate.conf** file contains a list of newline separated records, each of which is composed of a keyword and arguments, which are separated by white space. Lines beginning with "#" are treated as comments and ignored. However, a "#" in the middle of a line does not start comment.

The configuration options are as follows:

ignore *pattern ...*
Ignore files or directories. When building the database, do not descend into files or directories which match one of the specified patterns. The matched files or directories are not stored to the database.

Default: Not specified.

ignorecontents *pattern ...*
Ignore contents of directories. When building the database, do not descend into files or directories which match one of the specified patterns. The matched files or directories themselves are stored to the database.

Default: Not specified.

ignorefs *type ...*
Ignore file system by type, adding *type* to the default list. When building the database, do not descend into file systems which are of the specified *type*. The mount points are not stored to the database. If a "!" is prepended to *type*, the meaning is negated, that is, ignore file systems which do not have the type. As a special case, if "none" is specified for *type*, the **ignorefs** list is cleared and all file systems are traversed.

type is used as an argument to find(1) **-fstype**. The sysctl(8) command can be used to find out the types of file systems that are available on the system:

```
sysctl vfs.generic.fstypes
```

Default: !local cd9660 fdesc kernfs procfs

searchpath *directory ...*
Specify base directories to be put in the database.

Default: /

workdir *directory*
Specify the working directory of locate.updatedb, in which a temporary file is placed. The temporary file is a list of all files, and you should specify a directory that has enough space to hold it.

Default: /tmp

Refer to find(1) for the details of *pattern* (see **-path** expression) and *type* (see **-fstype** expression).

FILES

/etc/locate.conf The file **locate.conf** resides in /etc.

SEE ALSO

`find`(1), `locate`(1), `locate.updatedb`(8), `sysctl`(8)

HISTORY

The **`locate.conf`** file format first appeared in NetBSD 2.0.

AUTHORS

ITOH Yasufumi ⟨itohy@NetBSD.org⟩

NAME

login.access — login access control table

DESCRIPTION

The **login.access** file specifies on which ttys or from which hosts certain users are allowed to login.

At login, the /etc/login.access file is checked for the first entry that matches a specific user/host or user/tty combination. That entry can either allow or deny login access to that user.

Each entry have three fields separated by colon:

- The first field indicates the permission given if the entry matches. It can be either "+" (allow access) or "-" (deny access) .
- The second field is a comma separated list of users or groups for which the current entry applies. NIS netgroups can used (if configured) if preceeded by @. The magic string ALL matches all users. A group will match if the user is a member of that group, or it is the user's primary group.
- The third field is a list of ttys, or network names. A network name can be either a hostname, a domain (indicated by a starting period), or a netgroup. As with the user list, ALL matches anything. LOCAL matches a string not containing a period.

If the string EXCEPT is found in either the user or from list, the rest of the list are exceptions to the list before EXCEPT.

BUGS

If there's a user and a group with the same name, there is no way to make the group match if the user also matches.

SEE ALSO

login(1)

AUTHORS

The **login_access**() function was written by Wietse Venema. This manual page was written for Heimdal.

NAME

login.access — login access control table

DESCRIPTION

The **login.access** file specifies (user, host) combinations and/or (user, tty) combinations for which a login will be either accepted or refused.

When someone logs in, the **login.access** is scanned for the first entry that matches the (user, host) combination, or, in case of non-networked logins, the first entry that matches the (user, tty) combination. The permissions field of that table entry determines whether the login will be accepted or refused.

Each line of the login access control table has three fields separated by a ‘:’ character: *permission*:*users*:*origins*

The first field should be a "+" (access granted) or "-" (access denied) character. The second field should be a list of one or more login names, group names, or ALL (always matches). The third field should be a list of one or more tty names (for non-networked logins), host names, domain names (begin with "."), host addresses, internet network numbers (end with "."), ALL (always matches) or LOCAL (matches any string that does not contain a "." character). If you run NIS you can use @netgroupname in host or user patterns.

The EXCEPT operator makes it possible to write very compact rules.

The group file is searched only when a name does not match that of the logged-in user. Only groups are matched in which users are explicitly listed: the program does not look at a user’s primary group id value.

FILES

`/etc/login.access` The **login.access** file resides in `/etc`.

SEE ALSO

`login`(1), `pam`(8)

AUTHORS

Guido van Rooij

NAME

login.conf — login class capability data base

SYNOPSIS

login.conf

DESCRIPTION

The **login.conf** file describes the various attributes of login classes. A login class determines what styles of authentication are available as well as session resource limits and environment setup. While designed primarily for the login(1) program, it is also used by other programs, e.g., rexecd(8), which need to set up a user environment.

The class to be used is normally determined by the class field in the password file (see passwd(5)). The class is used to look up a corresponding entry in the login.conf file. A special class called “default” will be used (if it exists) if the field in the password file is empty.

CAPABILITIES

Refer to getcap(3) for a description of the file layout. An example entry is:

```
classname|Description entry:\
        :capability=value:\
        :booleancapability:\
                ...
        :lastcapability=value:
```

All entries in the **login.conf** file are either boolean or use a ‘=’ to separate the capability from the value. The types are described after the capability table.

Name	Type	Default	Description
copyright	file		File containing additional copyright information.
coredumpsize	size		Maximum coredump size limit.
cputime	time		CPU usage limit.
datasize	size		Maximum data size limit.
filesize	size		Maximum file size limit.
host.allow	string		A comma-separated list of host name or IP address patterns from which a class is allowed access. Access is instead denied from any hosts preceded by ‘!’. Patterns can contain the sh(1) -style ‘*’ and ‘?’ wildcards. The **host.deny** entry is checked before **host.allow**. (Currently used only by sshd(8).)
host.deny	string		A comma-separated list of host name or IP address patterns from which a class is denied access. Patterns as per **host.allow**, although a matched pattern that has been negated with ‘!’ is ignored. (Currently used only by sshd(8).)

hushlogin	bool	`false`	Same as having a `$HOME/.hushlogin` file. See `login`(1).
ignorenologin	bool	`false`	Not affected by `nologin` files.
login-retries	number	10	Maximum number of login attempts allowed.
login-backoff	number	3	Number of login attempts after which to start random back-off.
maxproc	number		Maximum number of process.
memorylocked	size		Maximum locked in core memory size limit.
memoryuse	size		Maximum in core memoryuse size limit.
minpasswordlen	number		The minimum length a local password may be. Used by the `passwd`(1) utility.
nologin	file		If the file exists it will be displayed and the login session will be terminated.
openfiles	number		Maximum number of open file descriptors per process.
passwordtime	time		Used by `passwd`(1) to set next password expiry date.
password-warn	time	`2w`	If the user's password will expire within this length of time then warn the user of this.
path	path	`/bin /usr/bin`	Default search path.
priority	number		Initial priority (nice) level.
requirehome	bool	`false`	Require home directory to login.
sbsize	size		Maximum socket buffer size limit.
setenv	list		Comma or whitespace separated list of environment variables and values to be set. Commas and whitespace can be escaped using \\.
shell	program		Session shell to execute rather than the shell specified in the password file. The `SHELL` environment variable will contain the shell specified in the password file.
stacksize	size		Maximum stack size limit.
tc	string		A "continuation" entry, which must be the last capability provided. More capabilities are read from the named entry. The capabilities given before **tc** override those in the entry invoked by **tc**.

term	string	`su`	Default terminal type if not able to determine from other means.
umask	number	`022`	Initial umask. Should always have a leading `0` to assure octal interpretation. See `umask`(2).
welcome	file	`/etc/motd`	File containing welcome message.

The resource limit entries (**coredumpsize**, **cputime**, **datasize**, **filesize**, **maxproc**, **memorylocked**, **memoryuse**, **openfiles**, **sbsize**, and **stacksize**) actually specify both the maximum and current limits (see `getrlimit`(2)). The current limit is the one normally used, although the user is permitted to increase the current limit to the maximum limit. The maximum and current limits may be specified individually by appending a '**−max**' or '**−cur**' to the capability name (e.g., **openfiles-max** and **openfiles-cur**).

NetBSD will never define capabilities which start with `x-` or `X-`, these are reserved for external use (unless included through contributed software).

The argument types are defined as:

bool If the name is present, then the boolean value is true; otherwise, it is false.

file Path name to a text file.

list A comma or whitespace separated list of values.

number A number. A leading `0x` implies the number is expressed in hexadecimal. A leading `0` implies the number is expressed in octal. Any other number is treated as decimal.

path A space separated list of path names. If a '`~`' is the first character in the path name, the '`~`' is expanded to the user's home directory.

program A path name to program.

size A number which expresses a size in bytes. It may have a trailing `b` to multiply the value by 512, a `k` to multiply the value by 1 K (1024), and a `m` to multiply the value by 1 M (1048576).

time A time in seconds. A time may be expressed as a series of numbers which are added together. Each number may have a trailing character to represent time units:

- **y** Indicates a number of 365 day years.
- **w** Indicates a number of 7 day weeks.
- **d** Indicates a number of 24 hour days.
- **h** Indicates a number of 60 minute hours.
- **m** Indicates a number of 60 second minutes.
- **s** Indicates a number of seconds.

For example, to indicate 1 and 1/2 hours, the following string could be used: `1h30m`.

FILES

`/etc/login.conf`	login class capability database
`/etc/login.conf.db`	hashed database built with `cap_mkdb`(1)

SEE ALSO

`cap_mkdb`(1), `login`(1), `getcap`(3), `login_cap`(3), `ttys`(5), `ftpd`(8), `sshd`(8)

HISTORY

The **login.conf** configuration file appeared in NetBSD 1.5.

NAME

magic — file command's magic number file

DESCRIPTION

This manual page documents the format of the magic file as used by the file(1) command, version 4.21. The file(1) command identifies the type of a file using, among other tests, a test for whether the file begins with a certain "magic number". The file /usr/share/misc/magic specifies what magic numbers are to be tested for, what message to print if a particular magic number is found, and additional information to extract from the file.

Each line of the file specifies a test to be performed. A test compares the data starting at a particular offset in the file with a 1-byte, 2-byte, or 4-byte numeric value or a string. If the test succeeds, a message is printed. The line consists of the following fields:

offset A number specifying the offset, in bytes, into the file of the data which is to be tested.

type The type of the data to be tested. The possible values are:

- byte — A one-byte value.
- short — A two-byte value (on most systems) in this machine's native byte order.
- long — A four-byte value (on most systems) in this machine's native byte order.
- quad — An eight-byte value (on most systems) in this machine's native byte order.
- string — A string of bytes. The string type specification can be optionally followed by /[Bbc]∗. The "B" flag compacts whitespace in the target, which must contain at least one whitespace character. If the magic has n consecutive blanks, the target needs at least n consecutive blanks to match. The "b" flag treats every blank in the target as an optional blank. Finally the "c" flag, specifies case insensitive matching: lowercase characters in the magic match both lower and upper case characters in the targer, whereas upper case characters in the magic, only much uppercase characters in the target.
- pstring — A pascal style string where the first byte is interpreted as the an unsigned length. The string is not NUL terminated.
- date — A four-byte value interpreted as a UNIX date.
- qdate — A eight-byte value interpreted as a UNIX date.
- ldate — A four-byte value interpreted as a UNIX-style date, but interpreted as local time rather than UTC.
- qldate — An eight-byte value interpreted as a UNIX-style date, but interpreted as local time rather than UTC.
- beshort — A two-byte value (on most systems) in big-endian byte order.
- belong — A four-byte value (on most systems) in big-endian byte order.
- bequad — An eight-byte value (on most systems) in big-endian byte order.
- bedate — A four-byte value (on most systems) in big-endian byte order, interpreted as a Unix date.
- beqdate — An eight-byte value (on most systems) in big-endian byte order, interpreted as a Unix date.

`beldate` A four-byte value (on most systems) in big-endian byte order, interpreted as a UNIX-style date, but interpreted as local time rather than UTC.

`beqldate` An eight-byte value (on most systems) in big-endian byte order, interpreted as a UNIX-style date, but interpreted as local time rather than UTC.

`bestring16` A two-byte unicode (UCS16) string in big-endian byte order.

`leshort` A two-byte value (on most systems) in little-endian byte order.

`lelong` A four-byte value (on most systems) in little-endian byte order.

`lequad` An eight-byte value (on most systems) in little-endian byte order.

`ledate` A four-byte value (on most systems) in little-endian byte order, interpreted as a UNIX date.

`leqdate` An eight-byte value (on most systems) in little-endian byte order, interpreted as a UNIX date.

`leldate` A four-byte value (on most systems) in little-endian byte order, interpreted as a UNIX-style date, but interpreted as local time rather than UTC.

`leqldate` An eight-byte value (on most systems) in little-endian byte order, interpreted as a UNIX-style date, but interpreted as local time rather than UTC.

`lestring16` A two-byte unicode (UCS16) string in little-endian byte order.

`melong` A four-byte value (on most systems) in middle-endian (PDP-11) byte order.

`medate` A four-byte value (on most systems) in middle-endian (PDP-11) byte order, interpreted as a UNIX date.

`meldate` A four-byte value (on most systems) in middle-endian (PDP-11) byte order, interpreted as a UNIX-style date, but interpreted as local time rather than UTC.

`regex` A regular expression match in extended POSIX regular expression syntax (much like egrep). The type specification can be optionally followed by /[cse]∗. The “c” flag makes the match case insensitive, while the “s” or “e” flags update the offset to the starting or ending offsets of the match (only one should be used). By default, regex does not update the offset. The regular expression is always tested against the first `N` lines, where `N` is the given offset, thus it is only useful for (single-byte encoded) text. ^ and $ will match the beginning and end of individual lines, respectively, not beginning and end of file.

`search` A literal string search starting at the given offset. It must be followed by `<number>` which specifies how many matches shall be attempted (the range). This is suitable for searching larger binary expressions with variable offsets, using \ escapes for special characters.

`default` This is intended to be used with the text `x` (which is always true) and a message that is to be used if there are no other matches.

The numeric types may optionally be followed by & and a numeric value, to specify that the value is to be AND’ed with the numeric value before any comparisons are done. Prepending a `u` to the type indicates that ordered comparisons should be unsigned.

`test` The value to be compared with the value from the file. If the type is numeric, this value is specified in C form; if it is a string, it is specified as a C string with the usual escapes permitted (e.g. \n for new-line).

Numeric values may be preceded by a character indicating the operation to be performed. It may be `=`, to specify that the value from the file must equal the specified value, `<`, to specify that the value from the file must be less than the specified value, `>`, to specify that the value from the file must be greater than the specified value, `&`, to specify that the value from the file must have set all of the bits that are set in the specified value, `^`, to specify that the value from the file must have clear any of the bits that are set in the specified value, or `~`, the value specified after is negated before tested. `x`, to specify that any value will match. If the character is omitted, it is assumed to be `=`. For all tests except *string* and *regex*, operation `!` specifies that the line matches if the test does *not* succeed.

Numeric values are specified in C form; e.g. `13` is decimal, `013` is octal, and `0x13` is hexadecimal.

For string values, the byte string from the file must match the specified byte string. The operators `=`, `<` and `>` (but not `&`) can be applied to strings. The length used for matching is that of the string argument in the magic file. This means that a line can match any string, and then presumably print that string, by doing *>\0* (because all strings are greater than the null string).

The special test *x* always evaluates to true. `message` The message to be printed if the comparison succeeds. If the string contains a `printf`(3) format specification, the value from the file (with any specified masking performed) is printed using the message as the format string. If the string begins with ‘‘\b’’, the message printed is the remainder of the string with no whitespace added before it: multiple matches are normally separated by a single space.

Some file formats contain additional information which is to be printed along with the file type or need additional tests to determine the true file type. These additional tests are introduced by one or more > characters preceding the offset. The number of > on the line indicates the level of the test; a line with no > at the beginning is considered to be at level 0. Tests are arranged in a tree-like hierarchy: If a the test on a line at level *n* succeeds, all following tests at level *n+1* are performed, and the messages printed if the tests succeed, untile a line with level *n* (or less) appears. For more complex files, one can use empty messages to get just the "if/then" effect, in the following way:

```
0      string   MZ
>0x18  leshort  <0x40   MS-DOS executable
>0x18  leshort  >0x3f   extended PC executable (e.g., MS Windows)
```

Offsets do not need to be constant, but can also be read from the file being examined. If the first character following the last > is a (then the string after the parenthesis is interpreted as an indirect offset. That means that the number after the parenthesis is used as an offset in the file. The value at that offset is read, and is used again as an offset in the file. Indirect offsets are of the form: *((x [.[bslBSL]][+−][y])*. The value of *x* is used as an offset in the file. A byte, short or long is read at that offset depending on the *[bslBSLm]* type specifier. The capitalized types interpret the number as a big endian value, whereas the small letter versions interpret the number as a little endian value; the *m* type interprets the number as a middle endian (PDP-11) value. To that number the value of *y* is added and the result is used as an offset in the file. The default type if one is not specified is long.

That way variable length structures can be examined:

```
# MS Windows executables are also valid MS-DOS executables
0           string  MZ
>0x18       leshort <0x40   MZ executable (MS-DOS)
# skip the whole block below if it is not an extended executable
>0x18       leshort >0x3f
>>(0x3c.l)  string  PE\0\0  PE executable (MS-Windows)
>>(0x3c.l)  string  LX\0\0  LX executable (OS/2)
```

This strategy of examining has one drawback: You must make sure that you eventually print something, or users may get empty output (like, when there is neither PE\0\0 nor LE\0\0 in the above example)

If this indirect offset cannot be used as-is, there are simple calculations possible: appending *[+-*/%&|^]<number>* inside parentheses allows one to modify the value read from the file before it is used as an offset:

```
# MS Windows executables are also valid MS-DOS executables
0           string  MZ
# sometimes, the value at 0x18 is less that 0x40 but there's still an
# extended executable, simply appended to the file
>0x18       leshort <0x40
>>(4.s*512) leshort 0x014c  COFF executable (MS-DOS, DJGPP)
>>(4.s*512) leshort !0x014c MZ executable (MS-DOS)
```

Sometimes you do not know the exact offset as this depends on the length or position (when indirection was used before) of preceding fields. You can specify an offset relative to the end of the last up-level field using '&' as a prefix to the offset:

```
0           string  MZ
>0x18       leshort >0x3f
>>(0x3c.l)  string  PE\0\0    PE executable (MS-Windows)
# immediately following the PE signature is the CPU type
>>>&0       leshort 0x14c     for Intel 80386
>>>&0       leshort 0x184     for DEC Alpha
```

Indirect and relative offsets can be combined:

```
0             string  MZ
>0x18         leshort <0x40
>>(4.s*512)   leshort !0x014c MZ executable (MS-DOS)
# if it's not COFF, go back 512 bytes and add the offset taken
# from byte 2/3, which is yet another way of finding the start
# of the extended executable
>>>&(2.s-514) string  LE      LE executable (MS Windows VxD driver)
```

Or the other way around:

```
0                string  MZ
>0x18            leshort >0x3f
>>(0x3c.l)       string  LE\0\0  LE executable (MS-Windows)
# at offset 0x80 (-4, since relative offsets start at the end
# of the up-level match) inside the LE header, we find the absolute
# offset to the code area, where we look for a specific signature
>>>(&0x7c.l+0x26) string  UPX     \b, UPX compressed
```

Or even both!

```
0               string  MZ
>0x18           leshort >0x3f
>>(0x3c.l)      string  LE\0\0 LE executable (MS-Windows)
# at offset 0x58 inside the LE header, we find the relative offset
# to a data area where we look for a specific signature
>>>&(&0x54.l-3)  string  UNACE  \b, ACE self-extracting archive
```

Finally, if you have to deal with offset/length pairs in your file, even the second value in a parenthesized expression can be taken from the file itself, using another set of parentheses. Note that this additional indi-

rect offset is always relative to the start of the main indirect offset.

```
0                  string       MZ
>0x18              leshort      >0x3f
>>(0x3c.l)         string       PE\0\0 PE executable (MS-Windows)
# search for the PE section called ".idata"...
>>>&0xf4           search/0x140 .idata
# ...and go to the end of it, calculated from start+length;
# these are located 14 and 10 bytes after the section name
>>>>(&0xe.l+(-4)) string       PK\3\4 \b, ZIP self-extracting archive
```

SEE ALSO

`file`(1) − the command that reads this file.

BUGS

The formats `long`, `belong`, `lelong`, `melong`, `short`, `beshort`, `leshort`, `date`, `bedate`, `medate`, `ledate`, `beldate`, `leldate`, and `meldate` are system-dependent; perhaps they should be specified as a number of bytes (2B, 4B, etc), since the files being recognized typically come from a system on which the lengths are invariant.

NAME

mailer.conf — configuration file for mailwrapper(8)

DESCRIPTION

The file /etc/mailer.conf contains a series of lines of the form

name program [*arguments ...*]

The first word of each line is the name of a program invoking mailwrapper(8). (For example, on a typical system /usr/sbin/sendmail would be a symbolic link to mailwrapper(8), as would newaliases(1) and mailq(1). Thus, name might be “sendmail” or “newaliases” etc.)

The second word of each line is the name of the program to actually execute when the first name is invoked.

The further *arguments*, if any, are passed to the program, followed by the arguments mailwrapper(8) was called with.

The file may also contain comment lines, denoted by a ‘#’ mark in the first column of any line.

The default mailer is sendmail(8), which will also start by default (unless specifically disabled via an rc.conf(5) setting) so that locally generated mail can be delivered, if the “sendmail” setting in /etc/mailer.conf is set to “/usr/libexec/sendmail/sendmail”.

FILES

/etc/mailer.conf

EXAMPLES

This example shows how to set up **mailer.conf** to invoke the traditional sendmail(8) program:

```
# Execute the "real" sendmail program located in
# /usr/libexec/sendmail/sendmail
sendmail        /usr/libexec/sendmail/sendmail
send-mail       /usr/libexec/sendmail/sendmail
mailq           /usr/libexec/sendmail/sendmail
newaliases      /usr/libexec/sendmail/sendmail
hoststat        /usr/libexec/sendmail/sendmail
purgestat       /usr/libexec/sendmail/sendmail
```

This example shows how to invoke the postfix(1) MTA suite in place of sendmail(8):

```
# Emulate sendmail using postfix
sendmail        /usr/libexec/postfix/sendmail
send-mail       /usr/libexec/postfix/sendmail
mailq           /usr/libexec/postfix/sendmail
newaliases      /usr/libexec/postfix/sendmail
```

This example shows the use of the mini-sendmail package from pkgsrc in place of sendmail(8): Note the use of additional arguments.

```
# Send outgoing mail to a smart relay using mini-sendmail
sendmail        /usr/pkg/sbin/mini-sendmail -srelayhost
send-mail       /usr/pkg/sbin/mini-sendmail -srelayhost
```

SEE ALSO

mail(1), mailq(1), newaliases(1), postfix(1), mailwrapper(8), sendmail(8)

`pkgsrc/mail/mini_sendmail`

HISTORY

`mailer.conf` appeared in NetBSD 1.4.

AUTHORS

Perry E. Metzger ⟨perry@piermont.com⟩

BUGS

The entire reason this program exists is a crock. Instead, a command for how to submit mail should be standardized, and all the "behave differently if invoked with a different name" behavior of things like `mailq`(1) should go away.

NAME

man.conf — configuration file for manual pages

DESCRIPTION

The **man.conf** file contains the default configuration used by man(1), apropos(1), whatis(1), catman(8), and makewhatis(8) to find manual pages and information about manual pages (e.g. the whatis database).

Manual pages are located by searching an ordered set of directories called the "man path" for a file that matches the name of the requested page. Each directory in the search path usually has a set of subdirectories in it (though this is not required). When subdirectories are used, there are normally two subdirectories for each section of the manual. One subdirectory contains formatted copies of that section's manual pages that can be directly displayed to a terminal, while the other section subdirectory contains unformatted copies of the pages (see nroff(1) and mdoc(7)). Formatted manual pages are normally named with a trailing ".0" suffix.

The **man.conf** file contains comment and configuration lines. Comment lines start with the "#" character. Blank lines are also treated as comment lines. Configuration lines consist of a configuration keyword followed by a configuration string. There are two types of configuration keywords: control keywords and section keywords. Control keywords must start with the "_" character. The following control keywords are currently defined:

_build identifies the set of suffixes used for manual pages that must be formatted for display and the command that should be used to format them. Manual file names, regardless of their format, are expected to end in a ".*" pattern, i.e. a "." followed by some suffix. The first field of a _build line contains a man page suffix specification. The suffix specification may contain the normal shell globbing characters (NOT including curly braces ("{}")). The rest of the _build line is a shell command line whose standard output is a formatted manual page that can be directly displayed to the user. Any occurrences of the string "%s" in the shell command line will be replaced by the name of the file which is being formatted.

_crunch used by catman(8) to determine how to crunch formatted pages which originally were compressed man pages: The first field lists a suffix which indicates what kind of compression were used to compress the man page. The rest of the line must be a shell command line, used to compress the formatted pages.

_default contains the system-wide default man path used to search for man pages.

_subdir contains the list (in search order) of section subdirectories which will be searched in any man path directory named with a trailing slash ("/") character. This list is also used, even if there is no trailing slash character, when a path is specified to the man(1) utility by the user, by the MANPATH environment variable, or by the **-M** and **-m** options.

_suffix identifies the set of suffixes used for formatted man pages (the ".0" suffix is normally used here). Formatted man pages can be directly displayed to the user. Each suffix may contain the normal shell globbing characters (NOT including curly braces ("{}")).

_version contains the version of the configuration file.

_whatdb defines the full pathname (not just a directory path) for a database to be used by the apropos(1) and whatis(1) commands. The pathname may contain the normal shell globbing characters, including curly braces ("{}"); to escape a shell globbing character, precede it with a backslash ("\").

Section configuration lines in **man.conf** consist of a section keyword naming the section and a configuration string that defines the directory or subdirectory path that the section's manual pages are located in. The path may contain the normal shell globbing characters, including curly braces ("{}"); to escape a shell

globbing character, precede it with a backslash (“\”). Section keywords must not start with the “_” character.

A section path may contain either a list of absolute directories or a list of or relative directories (but not both). Relative directory paths are treated as a list of subdirectories that are appended to the current man path directory being searched. Section configuration lines with absolute directory paths (starting with “/”) completely replace the current man search path directory with their content.

Section configuration lines with absolute directory paths ending with a trailing slash character are expected to contain subdirectories of manual pages, (see the keyword “_subdir” above). The “_subdir” subdirectory list is not applied to absolute section directories if there is no trailing slash.

In addition to the above rules, the man(1) command also always checks in each directory that it searches for a subdirectory with the same name as the current machine type. If the machine-specific directory is found, it is also searched. This allows the manual to contain machine-specific man pages. Note that the machine subdirectory does not need to be specified in the **man.conf** file.

Multiple specifications for all types of **man.conf** configuration lines are cumulative and the entries are used in the order listed in the file; multiple entries may be listed per line, as well.

FILES

`/etc/man.conf` Standard manual configuration file.

EXAMPLES

Given the following **man.conf** file:

```
_version        BSD.2
_subdir         cat[123]
_suffix         .0
_build          .[1-9]  nroff -man %s
_build          .tbl    tbl %s | nroff -man
_default        /usr/share/man/
sect3           /usr/share/man/{old/,}cat3
```

By default, the command “`man mktemp`” will search for “mktemp.<any_digit>” and “mktemp.tbl” in the directories “`/usr/share/man/cat1`”, “`/usr/share/man/cat2`”, and “`/usr/share/man/cat3`”. If on a machine of type “vax”, the subdirectory “vax” in each directory would be searched as well, before the directory was searched.

If “mktemp.tbl” was found first, the command “`tbl <manual page> | nroff -man`” would be run to build a man page for display to the user.

The command “`man sect3 mktemp`” would search the directories “`/usr/share/man/old/cat3`” and “`/usr/share/man/cat3`”, in that order, for the mktemp manual page. If a subdirectory with the same name as the current machine type existed in any of them, it would be searched as well, before each of them were searched.

SEE ALSO

`apropos`(1), `machine`(1), `man`(1), `whatis`(1), `whereis`(1), `fnmatch`(3), `glob`(3), `catman`(8), `makewhatis`(8)

NAME

map3270 − database for mapping ascii keystrokes into IBM 3270 keys

SYNOPSIS

map3270

DESCRIPTION

When emulating IBM-style 3270 terminals under UNIX (see *tn3270*(1)), a mapping must be performed between sequences of keys hit on a user's (ascii) keyboard, and the keys that are available on a 3270. For example, a 3270 has a key labeled **EEOF** which erases the contents of the current field from the location of the cursor to the end. In order to accomplish this function, the terminal user and a program emulating a 3270 must agree on what keys will be typed to invoke the **EEOF** function.

The requirements for these sequences are:

1) that the first character of the sequence be outside of the standard ascii printable characters;

2) that no sequence *be* an initial part of another (although sequences may *share* initial parts).

FORMAT

The file consists of entries for various keyboards. The first part of an entry lists the names of the keyboards which use that entry. These names will often be the same as in */usr/share/misc/termcap* (see *termcap*(5)); however, note that often the terminals from various termcap entries will all use the same *map3270* entry; for example, both 925 and 925vb (for 925 with visual bells) would probably use the same *map3270* entry. Additionally, there are occasions when the terminal type defines a window manager, and it will then be necessary to specify a keyboard name (via the **KEYBD** environment variable) as the name of the entry. After the names, separated by vertical bars ('|'), comes a left brace ('{'); the definitions; and, finally, a right brace ('}').

Each definition consists of a reserved keyword (see list below) which identifies the 3270 function (extended as defined below), followed by an equal sign ('='), followed by the various ways to generate this particular function, followed by a semi-colon (';'). Each way is a sequence of strings of *printable* ascii characters enclosed inside single quotes (''); various ways (alternatives) are separated by vertical bars ('|').

Inside the single quotes, a few characters are special. A caret ('^') specifies that the next character is the ''control'' character of whatever the character is. So, '^a' represents control-a, ie: hexadecimal 1 (note that '^A' would generate the same code). To generate **rubout** (DEL), one enters '^?'. To represent a control character inside a file requires using the caret to represent a control sequence; simply typing control-A will not work. Note: the ctrl-caret sequence (to generate a hexadecimal 1E) is represented as '^^' (not '^\^').

In addition to the caret, a letter may be preceded by a backslash ('\'). Since this has little effect for most characters, its use is usually not recommended. For the case of a single quote (''), the backslash prevents that single quote from terminating the string. For the case of a caret ('^'), the backslash prevents the caret from having its special meaning. To have the backslash be part of the string, it is necessary to place two backslashes ('\\') in the file.

In addition, the following characters are special:

'\E' means an escape character;
'\n' means newline;
'\t' means tab;
'\r' means carriage return.

It is not necessary for each character in a string to be enclosed within single quotes. '\E\E\E' means three escape characters.

Comments, which may appear anywhere on a line, begin with a hash mark ('#'), and terminate at the end of

that line. However, comments cannot begin inside a quoted string; a hash mark inside a quoted string has no special meaning.

3270 KEYS SUPPORTED

The following is the list of 3270 key names that are supported in this file. Note that some of the keys don't really exist on a 3270. In particular, the developers of this file have relied extensively on the work at the Yale University Computer Center with their 3270 emulator which runs in an IBM Series/1 front end. The following list corresponds closely to the functions that the developers of the Yale code offer in their product.

In the following list, the starred ("*") functions are not supported by *tn3270*(1). An unsupported function will cause *tn3270(1)* to send a (possibly visual) bell sequence to the user's terminal.

3270 Key Name	Functional description
(*)LPRT	local print
DP	dup character
FM	field mark character
CURSEL	cursor select
CENTSIGN	EBCDIC cent sign
RESHOW	redisplay the screen
EINP	erase input
EEOF	erase end of field
DELETE	delete character
INSRT	toggle insert mode
TAB	field tab
BTAB	field back tab
COLTAB	column tab
COLBAK	column back tab
INDENT	indent one tab stop
UNDENT	undent one tab stop
NL	new line
HOME	home the cursor
UP	up cursor
DOWN	down cursor
RIGHT	right cursor
LEFT	left cursor
SETTAB	set a column tab
DELTAB	delete a columntab
SETMRG	set left margin
SETHOM	set home position
CLRTAB	clear all column tabs
(*)APLON	apl on
(*)APLOFF	apl off
(*)APLEND	treat input as ascii
(*)PCON	xon/xoff on
(*)PCOFF	xon/xoff off
DISC	disconnect (suspend)
(*)INIT	new terminal type
(*)ALTK	alternative keyboard dvorak
FLINP	flush input
ERASE	erase last character
WERASE	erase last word
FERASE	erase field
SYNCH	we are in synch with the user

RESET reset key-unlock keyboard
MASTER_RESET reset, unlock and redisplay
(*)XOFF please hold output
(*)XON please give me output
ESCAPE enter telnet command mode
WORDTAB tab to beginning of next word
WORDBACKTAB tab to beginning of current/last word
WORDEND tab to end of current/next word
FIELDEND tab to last non-blank of current/next unprotected (writable) field.

PA1 program attention 1
PA2 program attention 2
PA3 program attention 3

CLEAR local clear of the 3270 screen
TREQ test request
ENTER enter key

PFK1 program function key 1
PFK2 program function key 2
etc. etc.
PFK36 program function key 36

A SAMPLE ENTRY

The following entry is used by tn3270(1) when unable to locate a reasonable version in the user's environment and in /usr/share/misc/map3270:

```
name {          # actual name comes from TERM variable
clear = '^z';
flinp = '^x';
enter = '^m';
delete = '^d' | '^?';   # note that '^?' is delete (rubout)
synch = '^r';
reshow = '^v';
eeof = '^e';
tab = '^i';
btab = '^b';
nl = '^n';
left = '^h';
right = '^l';
up = '^k';
down = '^j';
einp = '^w';
reset = '^t';
xoff = '^s';
xon = '^q';
escape = '^c';
ferase = '^u';
insrt = ' ';
# program attention keys
pa1 = '^p1'; pa2 = '^p2'; pa3 = '^p3';
# program function keys
pfk1 = '\E1'; pfk2 = '\E2'; pfk3 = '\E3'; pfk4 = '\E4';
pfk5 = '\E5'; pfk6 = '\E6'; pfk7 = '\E7'; pfk8 = '\E8';
```

```
pfk9 = '\E9'; pfk10 = '\E0'; pfk11 = '\E-'; pfk12 = '\E=';
pfk13 = '\E!'; pfk14 = '\E@'; pfk15 = '\E#'; pfk16 = '\E$';
pfk17 = '\E%'; pfk18 = '\E'; pfk19 = '\E&'; pfk20 = '\E*';
pfk21 = '\E('; pfk22 = '\E)'; pfk23 = '\E_'; pfk24 = '\E+';
}
```

IBM 3270 KEY DEFINITIONS FOR AN ABOVE DEFINITION

The charts below show the proper keys to emulate each 3270 function when using the default key mapping supplied with *tn3270*(1) and *mset*(1).

Command Keys	IBM 3270 Key	Default Key(s)
	Enter	RETURN
	Clear	control-z
Cursor Movement Keys		
	New Line	control-n or Home
	Tab	control-i
	Back Tab	control-b
	Cursor Left	control-h
	Cursor Right	control-l
	Cursor Up	control-k
	Cursor Down	control-j or LINE FEED
Edit Control Keys		
	Delete Char	control-d or RUB
	Erase EOF	control-e
	Erase Input	control-w
	Insert Mode	ESC Space
	End Insert	ESC Space
Program Function Keys		
	PF1	ESC 1
	PF2	ESC 2
	...	...
	PF10	ESC 0
	PF11	ESC -
	PF12	ESC =
	PF13	ESC !
	PF14	ESC @
	...	...
	PF24	ESC +
Program Attention Keys		
	PA1	control-p 1
	PA2	control-p 2
	PA3	control-p 3
Local Control Keys		
	Reset After Error	control-r
	Purge Input Buffer	control-x
	Keyboard Unlock	control-t
	Redisplay Screen	control-v
Other Keys		
	Erase current field	control-u

FILES

/usr/share/misc/map3270

SEE ALSO

tn3270(1), mset(1), *Yale ASCII Terminal Communication System II Program Description/Operator's Manual* (IBM SB30-1911)

AUTHOR

Greg Minshall

BUGS

Tn3270 doesn't yet understand how to process all the functions available in *map3270;* when such a function is requested *tn3270* will beep at you.

The definition of "word" (for "word erase", "word tab") should be a run-time option. Currently it is defined as the kernel tty driver defines it (strings of non-whitespace); more than one person would rather use the "vi" definition (strings of specials, strings of alphanumeric).

NAME

master − Postfix master process configuration file format

DESCRIPTION

The Postfix mail system is implemented by small number of (mostly) client commands that are invoked by users, and by a larger number of services that run in the background.

Postfix services are implemented by daemon processes. These run in the background under control of the **master**(8) process. The master.cf configuration file defines how a client program connects to a service, and what daemon program runs when a service is requested. Most daemon processes are short-lived and terminate after serving **max_use** clients, or after inactivity for **max_idle** or more units of time.

All daemons specified here must speak a Postfix-internal protocol. In order to execute non-Postfix software use the **local**(8), **pipe**(8) or **spawn**(8) services, or run the server under control by **inetd**(8) or equivalent.

After changing master.cf you must execute "**postfix reload**" to reload the configuration.

SYNTAX

The general format of the master.cf file is as follows:

- Each logical line defines a single Postfix service. Each service is identified by its name and type as described below. When multiple lines specify the same service name and type, only the last one is remembered. Otherwise, the order of master.cf service definitions does not matter.
- Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.
- A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

Each logical line consists of eight fields separated by whitespace. These are described below in the order as they appear in the master.cf file.

Where applicable a field of "-" requests that the built-in default value be used. For boolean fields specify "y" or "n" to override the default value.

Service name

The service name syntax depends on the service type as described next.

Service type

Specify one of the following service types:

inet The service listens on a TCP/IP socket and is accessible via the network.

The service name is specified as *host:port*, denoting the host and port on which new connections should be accepted. The host part (and colon) may be omitted. Either host or port may be given in symbolic form (host or service name) or in numeric form (IP address or port number). Host information may be enclosed inside "[]", but this form is not necessary.

Examples: a service named **127.0.0.1:smtp** or **::1:smtp** receives mail via the loopback interface only; and a service named **10025** accepts connections on TCP port 10025 via all interfaces configured with the **inet_interfaces** parameter.

Note: with Postfix version 2.2 and later specify "**inet_interfaces = loopback-only**" in main.cf, instead of hard-coding loopback IP address information in master.cf or in main.cf.

unix The service listens on a UNIX-domain socket and is accessible for local clients only.

The service name is a pathname relative to the Postfix queue directory (pathname controlled with the **queue_directory** configuration parameter in main.cf).

On Solaris systems the **unix** type is implemented with streams sockets.

fifo The service listens on a FIFO (named pipe) and is accessible for local clients only.

The service name is a pathname relative to the Postfix queue directory (pathname controlled with the **queue_directory** configuration parameter in main.cf).

Private (default: y)

Whether or not access is restricted to the mail system. Internet (type **inet**) services can't be private.

Unprivileged (default: y)

Whether the service runs with root privileges or as the owner of the Postfix system (the owner name is controlled by the **mail_owner** configuration variable in the main.cf file).

The **local**(8), **pipe**(8), **spawn**(8), and **virtual**(8) daemons require privileges.

Chroot (default: y)

Whether or not the service runs chrooted to the mail queue directory (pathname is controlled by the **queue_directory** configuration variable in the main.cf file).

Chroot should not be used with the **local**(8), **pipe**(8), **spawn**(8), and **virtual**(8) daemons. Although the **proxymap**(8) server can run chrooted, doing so defeats most of the purpose of having that service in the first place.

The files in the examples/chroot-setup subdirectory of the Postfix source archive show set up a Postfix chroot environment on a variety of systems. See also BASIC_CONFIGURATION_README for issues related to running daemons chrooted.

Wake up time (default: 0)

Automatically wake up the named service after the specified number of seconds. The wake up is implemented by connecting to the service and sending a wake up request. A ? at the end of the wake-up time field requests that no wake up events be sent before the first time a service is used. Specify 0 for no automatic wake up.

The **pickup**(8), **qmgr**(8) and **flush**(8) daemons require a wake up timer.

Process limit (default: $default_process_limit)

The maximum number of processes that may execute this service simultaneously. Specify 0 for no process count limit.

NOTE: Some Postfix services must be configured as a single-process service (for example, **qmgr**(8)) and some services must be configured with no process limit (for example, **cleanup**(8)). These limits must not be changed.

Command name + arguments

The command to be executed. Characters that are special to the shell such as ">" or "|" have no special meaning here, and quotes cannot be used to protect arguments containing whitespace.

The command name is relative to the Postfix daemon directory (pathname is controlled by the **daemon_directory** configuration variable).

The command argument syntax for specific commands is specified in the respective daemon manual page.

The following command-line options have the same effect for all daemon programs:

-D Run the daemon under control by the command specified with the **debugger_command** variable in the main.cf configuration file. See DEBUG_README for hints and tips.

-o *name=value*
Override the named main.cf configuration parameter. The parameter value can refer to other parameters as *$name* etc., just like in main.cf. See **postconf**(5) for syntax.

NOTE 1: do not specify whitespace around the "=". In parameter values, either avoid whitespace altogether, use commas instead of spaces, or consider overrides like "-o name=$override_parameter" with $override_parameter set in main.cf.

NOTE 2: Over-zealous use of parameter overrides makes the Postfix configuration hard to understand and maintain. At a certain point, it might be easier to configure multiple instances of Postfix, instead of configuring multiple personalities via master.cf.

-v Increase the verbose logging level. Specify multiple **-v** options to make a Postfix daemon process increasingly verbose.

SEE ALSO

master(8), process manager
postconf(5), configuration parameters

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
BASIC_CONFIGURATION_README, basic configuration
DEBUG_README, Postfix debugging

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Initial version by
Magnus Baeck
Lund Institute of Technology
Sweden

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

mech, **qop** — GSS-API Mechanism and QOP files

SYNOPSIS

`/etc/gss/mech` `/etc/gss/qop`

DESCRIPTION

The `/etc/gss/mech` file contains a list of installed GSS-API security mechanisms. Each line of the file either contains a comment if the first character is '#' or it contains five fields with the following meanings:

Name — The name of this GSS-API mechanism.

Object identifier
: The OID for this mechanism.

Library — A shared library containing the implementation of this mechanism.

Kernel module (optional)
: A kernel module containing the implementation of this mechanism (not yet supported in FreeBSD).

Library options (optional)
: Optionsal parameters interpreted by the mechanism. Library options must be enclosed in brackets ([ ]) to differentiate them from the optional kernel module entry.

The `/etc/gss/qop` file contains a list of Quality of Protection values for use with GSS-API. Each line of the file either contains a comment if the first character is '#' or it contains three fields with the following meanings:

QOP string — The name of this Quality of Protection algorithm.

QOP value — The numeric value used to select this algorithm for use with GSS-API functions such as `gss_get_mic`(3).

Mechanism name
: The GSS-API mechanism name that corresponds to this algorithm.

EXAMPLES

This is a typical entry from `/etc/gss/mech`:

```
kerberosv5     1.2.840.113554.1.2.2   /usr/lib/libgssapi_krb5.so.8  -
```

This is a typical entry from `/etc/gss/qop`:

```
GSS_KRB5_CONF_C_QOP_DES              0x0100  kerberosv5
```

HISTORY

The **mech** manual page example first appeared in FreeBSD 7.0.

AUTHORS

This manual page was written by Doug Rabson ⟨dfr@FreeBSD.org⟩.

NAME

mixerctl.conf — audio mixer configuration file

SYNOPSIS

mixerctl.conf

DESCRIPTION

The /etc/mixerctl.conf file consists of mixerctl(1) variables to set at boot time. Each line of **mixerctl.conf** has the following format:

```
variable=value
```

To generate a **mixerctl.conf** from the current mixer settings, execute:

```
mixerctl -a > /etc/mixerctl.conf
```

Set **mixerctl** to YES in rc.conf(5) to have the variables set at boot time. Additionally, you can have the settings saved and restored for the devices of your choice by listing them in **mixerctl_mixers** in rc.conf(5).

FILES

/etc/mixerctl.conf

EXAMPLES

Example mixer settings for an esa(4) audio adapter:

```
outputs.master=255,255
outputs.master.mute=off
outputs.mono=255
outputs.mono.mute=on
outputs.mono.source=mixerout
outputs.headphones=255,255
outputs.headphones.mute=off
outputs.tone=255,255
inputs.speaker=255
inputs.speaker.mute=off
inputs.phone=191
inputs.phone.mute=on
inputs.mic=191
inputs.mic.mute=on
inputs.mic.preamp=off
inputs.mic.source=mic0
inputs.line=191,191
inputs.line.mute=on
inputs.cd=191,191
inputs.cd.mute=on
inputs.video=255,255
inputs.video.mute=off
inputs.aux=255,255
inputs.aux.mute=off
inputs.dac=191,191
inputs.dac.mute=off
record.source=mic
record.volume=255,255
record.volume.mute=off
```

```
record.mic=0
record.mic.mute=off
outputs.loudness=off
outputs.spatial=off
outputs.spatial.center=0
outputs.spatial.depth=0
```

SEE ALSO

`mixerctl`(1), `rc.conf`(5)

HISTORY

The **`mixerctl.conf`** configuration file first appeared in NetBSD 2.0.

NAME

mk.conf — make configuration file

DESCRIPTION

The **mk.conf** file overrides various parameters used during the build of the system.

Listed below are the **mk.conf** variables that may be set, the values to which each may be set, a brief description of what each variable does, and a reference to relevant manual pages.

NetBSD System variables

NETBSDSRCDIR

The path to the top level of the NetBSD sources. If make(1) is run from within the NetBSD source tree, the default is the top level of that tree (as determined by the presence of build.sh and tools/), otherwise **BSDSRCDIR** will be used.

BSDOBJDIR The real path to the 'obj' tree for the NetBSD source tree.

Default: /usr/obj

BSDSRCDIR The real path to the NetBSD source tree.

Default: /usr/src

BUILD If defined, 'make install' checks that the targets in the source directories are up-to-date and re-makes them if they are out of date, instead of blindly trying to install out of date or non-existent targets.

Default: Unset.

BUILDID Identifier for the build. The identifier will be appended to object directory names, and can be consulted in the make(1) configuration file in order to set additional build parameters, such as compiler flags.

Default: Unset.

COPTS Extra options for the C compiler. Should be appended to (e.g., **COPTS+=-g**), rather than explicitly set. Note that **CPUFLAGS**, not **COPTS**, should be used for compiler flags that select CPU-related options. Also note that **CFLAGS** should never be set in **mk.conf**.

CPUFLAGS Additional flags passed to the compiler/assembler to select CPU instruction set options, CPU tuning options, etc. Such options should not be specified in **COPTS**, because some parts of the build process need to override CPU-related compiler options.

DESTDIR Directory to contain the built NetBSD system. If set, special options are passed to the compilation tools to prevent their default use of the host system's **/usr/include**, **/usr/lib**, and so forth. This pathname should *not* end with a slash (/) character (for installation into the system's root directory, set **DESTDIR** to an empty string). The directory must reside on a file system which supports long file names and hard links.

Default: Empty string if **USETOOLS** is "yes"; unset otherwise.

Note: **build.sh** will provide a default of destdir.**MACHINE** (in the top-level **.OBJDIR**) unless run in 'expert' mode

MKBFD Can be set to "yes" or "no". Indicates whether **libbfd**, **libiberty**, or any of the things that depend upon them (such as the binutils, as(1), gdb(1), ld(1), dbsym(8), or mdsetimage(8)) should be built.

Default: “yes”

MKCATPAGES Can be set to “yes” or “no”. Indicates whether preformatted plaintext manual pages will be created during a build.

Default: “yes”

MKCRYPTO Can be set to “yes” or “no”. Indicates whether cryptographic code will be included in a build; provided for the benefit of countries that do not allow strong cryptography. Will not affect use of the standard low-security password encryption system, `crypt`(3).

Default: “yes”

If “no”, acts as **MKKERBEROS=no**.

MKCRYPTO_IDEA
Can be set to “yes” or “no”. Indicates whether IDEA support will be built into **libcrypto_idea.a**.

Default: “no”

MKCRYPTO_MDC2
Can be set to “yes” or “no”. Indicates whether MDC2 support will be built into **libcrypto_mdc2.a**.

Default: “no”

MKCRYPTO_RC5
Can be set to “yes” or “no”. Indicates whether RC5 support will be built into **libcrypto_rc5.a**.

Default: “no”

MKCVS Can be set to “yes” or “no”. Indicates whether `cvs`(1) is built.

Default: “yes”

MKDEBUG Can be set to “yes” or “no”. Indicates whether separate debugging symbols should be installed into **DESTDIR**`/usr/libdata/debug`.

Default: “no”

MKDEBUGLIB Can be set to “yes” or “no”. Indicates whether debug libraries (**lib**∗**_g.a**) will be built and installed during a build. Debug libraries are compiled with “`-g -DDEBUG`”.

Default: “no”

MKDOC Can be set to “yes” or “no”. Indicates whether system documentation destined for **DESTDIR**`/usr/share/doc` will be installed during a build.

Default: “yes”

MKDYNAMICROOT
Can be set to “yes” or “no”. Indicates whether all programs should be dynamically linked, and to install shared libraries required by `/bin` and `/sbin` and the shared linker `ld.elf_so`(1) into `/lib`. If ‘no’, link programs in `/bin` and `/sbin` statically.

MKGCC Can be set to “yes” or “no”. Indicates whether `gcc`(1) or any related libraries (**libg2c**, **libgcc**, **libobjc**, **libstdc+**) are built.

Default: “yes”

MKGDB Can be set to “yes” or “no”. Indicates whether `gdb`(1) is built.

Default: “yes”

MKHESIOD Can be set to “yes” or “no”. Indicates whether the Hesiod infrastructure (libraries and support programs) is built.

Default: “yes”

MKHOSTOBJ Can be set to “yes” or “no”. If set to “yes”, then for programs intended to be run on the compile host, the name, release, and architecture of the host operating system will be suffixed to the name of the object directory created by “make obj”. (This allows multiple host systems to compile NetBSD for a single target.) If set to “no”, then programs built to be run on the compile host will use the same object directory names as programs built to be run on the target.

Default: “no”

MKHTML Can be set to “yes” or “no”. Indicates whether the html manual pages are built and installed.

Default: “yes”

MKIEEEFP Can be set to “yes” or “no”. Indicates whether code for IEEE754/IEC60559 conformance is built. Has no effect on most platforms.

Default: “yes”

MKINFO Can be set to “yes” or “no”. Indicates whether GNU Info files, used for the documentation for most of the compilation tools, will be created and installed during a build.

Default: “yes”

MKIPFILTER Can be set to “yes” or “no”. Indicates whether the `ipf`(4) programs, headers and LKM will be compiled and installed during a build.

Default: “yes”

MKKERBEROS Can be set to “yes” or “no”. Indicates whether the Kerberos v5 infrastructure (libraries and support programs) is built.

Default: “yes”

MKLINKLIB Can be set to “yes” or “no”. Indicates whether all of the shared library infrastructure is built. If ‘no’, prevents: installation of the ∗**.a** libraries, installation of the ∗**_pic.a** libraries on PIC systems, building of ∗**.a** libraries on PIC systems, or installation of **.so** symlinks on ELF systems.

Default: “yes”

If “no”, acts as **MKPICINSTALL=no MKPROFILE=no**.

MKLINT Can be set to “yes” or “no”. Indicates whether `lint`(1) will be run against portions of the NetBSD source code during the build, and whether lint libraries will be installed into **DESTDIR**`/usr/libdata/lint`.

Default: “yes”

MKMAN Can be set to “yes” or “no”. Indicates whether manual pages will be installed during a build.

Default: “yes”

If “no”, acts as **MKCATPAGES=no MKHTML=no**.

MKMANZ Can be set to “yes” or “no”. Indicates whether manual pages should be compressed with `gzip`(1) at installation time.

Default: “no”

MKMODULAR Can be set to “yes” or “no”. Indicates whether support for the new kernel modules framework should be built. This will install new versions of `modload`(8), `modstat`(8) and `modunload`(8) that will not work with the ones used to manage LKMs. You will also need a kernel built with **`options MODULAR`** for this to be useful.

Default: “no”

MKNLS Can be set to “yes” or “no”. Indicates whether Native Language System (NLS) locale zone files will be compiled and installed during a build.

Default: “yes”

MKOBJ Can be set to “yes” or “no”. Indicates whether object directories will be created when running “make obj”. If set to “no”, then all built files will be located inside the regular source tree.

Default: “yes”

If “no”, acts as **MKOBJDIRS=no**.

MKOBJDIRS Can be set to “yes” or “no”. Indicates whether object directories will be created automatically (via a “make obj” pass) at the start of a build.

Default: “no”

MKPAM Can be set to “yes” or “no”. Indicates whether the `pam`(8) framework (libraries and support files) is built. The pre-PAM code is not supported and may be removed in the future.

Default: “yes”

MKPF Can be set to “yes” or “no”. Indicates whether the `pf`(4) programs, headers and LKM will be compiled and installed during a build.

Default: “yes”

MKPIC Can be set to “yes” or “no”. Indicates whether shared objects and libraries will be created and installed during a build. If set to “no”, the entire built system will be statically linked.

Default: Platform dependent. As of this writing, all platforms except **m68000** and **sh3** default to “yes”.

If “no”, acts as **MKPICLIB=no**.

MKPICINSTALL Can be set to “yes” or “no”. Indicates whether the `ar`(1) format libraries (**lib∗_pic.a**), used to generate shared libraries, are installed during a build.

Default: “yes”

MKPICLIB Can be set to “yes” or “no”. Indicates whether the `ar`(1) format libraries (**lib∗_pic.a**), used to generate shared libraries.

Default: “yes”

MKPOSTFIX Can be set to “yes” or “no”. Indicates whether Postfix is built.

Default: “yes”

MKPROFILE Can be set to “yes” or “no”. Indicates whether profiled libraries (**lib*_p.a**) will be built and installed during a build.

Default: “yes”; however, some platforms turn off **MKPROFILE** by default at times due to toolchain problems with profiled code.

MKSHARE Can be set to “yes” or “no”. Indicates whether files destined to reside in **DESTDIR**`/usr/share` will be built and installed during a build.

Default: “yes”

If “no”, acts as **MKCATPAGES=no MKDOC=no MKINFO=no MKHTML=no MKMAN=no MKNLS=no**.

MKSKEY Can be set to “yes” or “no”. Indicates whether the S/key infrastructure (libraries and support programs) is built.

Default: “yes”

MKSOFTFLOAT Can be set to “yes” or “no”. Indicates whether the compiler generates output containing library calls for floating point and possibly soft-float library support.

Default: “no”

MKUNPRIVED Can be set to “yes” or “no”. Indicates whether an unprivileged install will occur. The user, group, permissions, and file flags, will not be set on the installed item; instead the information will be appended to a file called `METALOG` in **DESTDIR**. The contents of `METALOG` is used during the generation of the distribution tar files to ensure that the appropriate file ownership is stored.

Default: “no”

MKUPDATE Can be set to “yes” or “no”. Indicates whether all install operations intended to write to **DESTDIR** will compare file timestamps before installing, and skip the install phase if the destination files are up-to-date. This also has implications on full builds (see next subsection).

Default: “no”

MKYP Can be set to “yes” or “no”. Indicates whether the YP (NIS) infrastructure (libraries and support programs) is built.

Default: “yes”

OBJMACHINE If defined, creates objdirs of the form `obj.`**MACHINE**, where **MACHINE** is the current architecture (as per ‘uname -m’).

RELEASEDIR If set, specifies the directory to which a `release`(7) layout will be written at the end of a “make release”.

Default: Unset.

Note: **build.sh** will provide a default of `releasedir` (in the top-level **.OBJDIR**) unless run in ‘expert’ mode

TOOLDIR Directory to hold the host tools, once built. This directory should be unique to a given host system and NetBSD source tree. (However, multiple targets may share the same **TOOLDIR**; the target-dependent files have unique names.) If unset, a default based on the `uname`(1) information of the host platform will be created in the **.OBJDIR** of `src`.

Default: Unset.

USETOOLS Indicates whether the tools specified by **TOOLDIR** should be used as part of a build in progress. Must be set to "yes" if cross-compiling.

yes Use the tools from **TOOLDIR**.

no Do not use the tools from **TOOLDIR**, but refuse to build native compilation tool components that are version-specific for that tool.

never Do not use the tools from **TOOLDIR**, even when building native tool components. This is similar to the traditional NetBSD build method, but does *not* verify that the compilation tools in use are up-to-date enough in order to build the tree successfully. This may cause build or runtime problems when building the whole NetBSD source tree.

Default: "yes" if building all or part of a whole NetBSD source tree (detected automatically); "no" otherwise (to preserve traditional semantics of the ⟨bsd.∗.mk⟩ `make`(1) include files).

pkgsrc system variables

Please see the pkgsrc guide at `http://www.netbsd.org/Documentation/pkgsrc/` or `pkgsrc/doc/pkgsrc.txt` for more variables used internally by the package system and `${PKGSRCDIR}/mk/defaults/mk.conf` for package-specific examples.

FILES

`/etc/mk.conf` This file.

`${PKGSRCDIR}/mk/defaults/mk.conf`
Examples for settings regarding the pkgsrc collection.

SEE ALSO

`make`(1), `/usr/share/mk/bsd.README`, `pkgsrc/doc/pkgsrc.txt`, `http://www.netbsd.org/Documentation/pkgsrc/`

HISTORY

The **`mk.conf`** file appeared in NetBSD 1.2.

NAME

moduli — system moduli file

DESCRIPTION

The /etc/moduli file contains the system-wide Diffie-Hellman prime moduli for sshd(8).

Each line in this file contains the following fields: Time, Type, Tests, Tries, Size, Generator, Modulus. The fields are separated by white space (tab or blank).

Time: *yyyymmddhhmmss*. Specifies the system time that the line was appended to the file. The value 00000000000000 means unknown (historic).

Type: *decimal*. Specifies the internal structure of the prime modulus.

0: unknown; often learned from peer during protocol operation, and saved for later analysis.
1: unstructured; a common large number.
2: safe (p = 2q + 1); meets basic structural requirements.
3: Schnorr.
4: Sophie-Germaine (q = (p-1)/2); usually generated in the process of testing safe or strong primes.
5: strong; useful for RSA public key generation.

Tests: *decimal (bit field)*. Specifies the methods used in checking for primality. Usually, more than one test is used.

0: not tested; often learned from peer during protocol operation, and saved for later analysis.
1: composite; failed one or more tests. In this case, the highest bit specifies the test that failed.
2: sieve; checked for division by a range of smaller primes.
4: Miller-Rabin.
8: Jacobi.
16: Elliptic Curve.

Tries: *decimal*. Depends on the value of the highest valid Test bit, where the method specified is:

0: not tested (always zero).
1: composite (irrelevant).
2: sieve; number of primes sieved. Commonly on the order of 32,000,000.
4: Miller-Rabin; number of M-R iterations. Commonly on the order of 32 to 64.
8: Jacobi; unknown (always zero).
16: Elliptic Curve; unused (always zero).

Size: *decimal*. Specifies the number of the most significant bit (0 to M).

Generator: *hex string*. Specifies the best generator for a Diffie-Hellman exchange. 0 = unknown or variable, 2, 3, 5, etc.

Modulus: *hex string*. The prime modulus.

The file should be searched for moduli that meet the appropriate Time, Size and Generator criteria. When more than one meet the criteria, the selection should be weighted toward newer moduli, without completely disqualifying older moduli.

Note that sshd(8) uses only the Size criteria and then selects a modulus at random if more than one meet the Size criteria.

FILES

`/etc/moduli`

SEE ALSO

`qsieve`(1), `sshd`(8)

HISTORY

The **`moduli`** file appeared in OpenBSD 2.8 and NetBSD 1.6.

NAME

monthly.conf — monthly maintenance configuration file

DESCRIPTION

The **monthly.conf** file specifies which of the standard /etc/monthly services are performed. The /etc/monthly script is normally run in the morning of the 1st of every month, on a NetBSD system. Note: this is disabled (commented out) in the default root's crontab(5) file.

There are currently no monthly tasks.

FILES

/etc/monthly	monthly maintenance script
/etc/monthly.conf	monthly maintenance configuration
/etc/monthly.local	local site additions to /etc/monthly

SEE ALSO

daily.conf(5), weekly.conf(5)

HISTORY

The **monthly.conf** file appeared in NetBSD 1.3.

NAME

motd — file containing message(s) of the day

DESCRIPTION

The file `/etc/motd` is normally displayed by `login`(1) after a user has logged in but before the shell is run. It is generally used for important system-wide announcements. During system startup, a line containing the kernel version string is prepended to this file.

Individual users may suppress the display of this file by creating a file named "`.hushlogin`" in their home directories.

FILES

`/etc/motd`

EXAMPLES

```
NetBSD 1.0 (SUN_LAMP) #9: Sun Nov 20 22:47:57 PST 1994

Make sure you have a .forward file...

4/17   Machine will be down for backups all day Saturday.
```

SEE ALSO

`login`(1)

NAME

mtree — format of mtree dir hierarchy files

DESCRIPTION

The **mtree** format is a textual format that describes a collection of filesystem objects. Such files are typically used to create or verify directory hierarchies.

General Format

An **mtree** file consists of a series of lines, each providing information about a single filesystem object. Leading whitespace is always ignored.

When encoding file or pathnames, any backslash character or character outside of the 95 printable ASCII characters must be encoded as a a backslash followed by three octal digits. When reading mtree files, any appearance of a backslash followed by three octal digits should be converted into the corresponding character.

Each line is interpreted independently as one of the following types:

Signature — The first line of any mtree file must begin with "#mtree". If a file contains any full path entries, the first line should begin with "#mtree v2.0", otherwise, the first line should begin with "#mtree v1.0".

Blank — Blank lines are ignored.

Comment — Lines beginning with **#** are ignored.

Special — Lines beginning with **/** are special commands that influence the interpretation of later lines.

Relative — If the first whitespace-delimited word has no **/** characters, it is the name of a file in the current directory. Any relative entry that describes a directory changes the current directory.

dot-dot — As a special case, a relative entry with the filename **..** changes the current directory to the parent directory. Options on dot-dot entries are always ignored.

Full — If the first whitespace-delimited word has a **/** character after the first character, it is the pathname of a file relative to the starting directory. There can be multiple full entries describing the same file.

Some tools that process **mtree** files may require that multiple lines describing the same file occur consecutively. It is not permitted for the same file to be mentioned using both a relative and a full file specification.

Special commands

Two special commands are currently defined:

/set — This command defines default values for one or more keywords. It is followed on the same line by one or more whitespace-separated keyword definitions. These definitions apply to all following files that do not specify a value for that keyword.

/unset — This command removes any default value set by a previous **/set** command. It is followed on the same line by one or more keywords separated by whitespace.

Keywords

After the filename, a full or relative entry consists of zero or more whitespace-separated keyword definitions. Each such definition consists of a key from the following list immediately followed by an '=' sign and a value. Software programs reading mtree files should warn about unrecognized keywords.

Currently supported keywords are as follows:

cksum The checksum of the file using the default algorithm specified by the `cksum`(1) utility.

contents The full pathname of a file that holds the contents of this file.

flags The file flags as a symbolic name. See `chflags`(1) for information on these names. If no flags are to be set the string "none" may be used to override the current default.

gid The file group as a numeric value.

gname The file group as a symbolic name.

ignore Ignore any file hierarchy below this file.

link The target of the symbolic link when type=link.

md5 The MD5 message digest of the file.

md5digest
A synonym for **md5**.

mode The current file's permissions as a numeric (octal) or symbolic value.

nlink The number of hard links the file is expected to have.

nochange Make sure this file or directory exists but otherwise ignore all attributes.

ripemd160digest
The RIPEMD160 message digest of the file.

rmd160 A synonym for **ripemd160digest**.

rmd160digest
A synonym for **ripemd160digest**.

sha1 The FIPS 160-1 ("SHA-1") message digest of the file.

sha1digest
A synonym for **sha1**.

sha256 The FIPS 180-2 ("SHA-256") message digest of the file.

sha256digest
A synonym for **sha256**.

size The size, in bytes, of the file.

time The last modification time of the file.

type The type of the file; may be set to any one of the following:

block	block special device
char	character special device
dir	directory
fifo	fifo
file	regular file
link	symbolic link
socket	socket

uid The file owner as a numeric value.

uname The file owner as a symbolic name.

SEE ALSO

`cksum`(1), `find`(1), `mtree`(8)

BUGS

The FreeBSD implementation of mtree does not currently support the **`mtree`** 2.0 format. The requirement for a “#mtree” signature line is new and not yet widely implemented.

HISTORY

The **`mtree`** utility appeared in 4.3BSD−Reno. The MD5 digest capability was added in FreeBSD 2.1, in response to the widespread use of programs which can spoof `cksum`(1). The SHA-1 and RIPEMD160 digests were added in FreeBSD 4.0, as new attacks have demonstrated weaknesses in MD5. The SHA-256 digest was added in FreeBSD 6.0. Support for file flags was added in FreeBSD 4.0, and mostly comes from NetBSD. The “full” entry format was added by NetBSD.

NAME

mysql_table − Postfix MySQL client configuration

SYNOPSIS

postmap -q "*string***" mysql:/etc/postfix/filename**

postmap -q - mysql:/etc/postfix/*filename* <*inputfile*

DESCRIPTION

The Postfix mail system uses optional tables for address rewriting or mail routing. These tables are usually in **dbm** or **db** format.

Alternatively, lookup tables can be specified as MySQL databases. In order to use MySQL lookups, define a MySQL source as a lookup table in main.cf, for example:

alias_maps = mysql:/etc/mysql-aliases.cf

The file /etc/postfix/mysql-aliases.cf has the same format as the Postfix main.cf file, and can specify the parameters described below.

BACKWARDS COMPATIBILITY

For compatibility with other Postfix lookup tables, MySQL parameters can also be defined in main.cf. In order to do that, specify as MySQL source a name that doesn't begin with a slash or a dot. The MySQL parameters will then be accessible as the name you've given the source in its definition, an underscore, and the name of the parameter. For example, if the map is specified as "mysql:*mysqlname*", the parameter "hosts" below would be defined in main.cf as "*mysqlname*_hosts".

Note: with this form, the passwords for the MySQL sources are written in main.cf, which is normally world-readable. Support for this form will be removed in a future Postfix version.

Postfix 2.2 has enhanced query interfaces for MySQL and PostgreSQL, these include features previously available only in the Postfix LDAP client. In the new interface the SQL query is specified via a single **query** parameter (described in more detail below). When the new **query** parameter is not specified in the map definition, Postfix reverts to the old interface, with the SQL query constructed from the **select_field**, **table**, **where_field** and **additional_conditions** parameters. The old interface will be gradually phased out. To migrate to the new interface set:

query = SELECT [*select_field*]
FROM [*table*]
WHERE [*where_field*] = '%s'
[*additional_conditions*]

Insert the value, not the name, of each legacy parameter. Note that the **additional_conditions** parameter is optional and if not empty, will always start with **AND**.

LIST MEMBERSHIP

When using SQL to store lists such as $mynetworks, $mydestination, $relay_domains, $local_recipient_maps, etc., it is important to understand that the table must store each list member as a separate key. The table lookup verifies the *existence* of the key. See "Postfix lists versus tables" in the DATABASE_README document for a discussion.

Do NOT create tables that return the full list of domains in $mydestination or $relay_domains etc., or IP addresses in $mynetworks.

DO create tables with each matching item as a key and with an arbitrary value. With SQL databases it is not uncommon to return the key itself or a constant value.

MYSQL PARAMETERS

hosts The hosts that Postfix will try to connect to and query from. Specify *unix:* for UNIX domain sockets, *inet:* for TCP connections (default). Example:

hosts = host1.some.domain host2.some.domain
hosts = unix:/file/name

The hosts are tried in random order, with all connections over UNIX domain sockets being tried before those over TCP. The connections are automatically closed after being idle for about 1 minute, and are re-opened as necessary. Postfix versions 2.0 and earlier do not randomize the host order.

NOTE: if you specify localhost as a hostname (even if you prefix it with *inet:*), MySQL will connect to the default UNIX domain socket. In order to instruct MySQL to connect to localhost over TCP you have to specify

hosts = 127.0.0.1

user, password

The user name and password to log into the mysql server. Example:

user = someone
password = some_password

dbname

The database name on the servers. Example:

dbname = customer_database

query The SQL query template used to search the database, where **%s** is a substitute for the address Postfix is trying to resolve, e.g.

query = SELECT replacement FROM aliases WHERE mailbox = '%s'

This parameter supports the following '%' expansions:

%% This is replaced by a literal '%' character.

%s This is replaced by the input key. SQL quoting is used to make sure that the input key does not add unexpected metacharacters.

%u When the input key is an address of the form user@domain, **%u** is replaced by the SQL quoted local part of the address. Otherwise, **%u** is replaced by the entire search string. If the localpart is empty, the query is suppressed and returns no results.

%d When the input key is an address of the form user@domain, **%d** is replaced by the SQL quoted domain part of the address. Otherwise, the query is suppressed and returns no results.

%[SUD]

The upper-case equivalents of the above expansions behave in the **query** parameter identically to their lower-case counter-parts. With the **result_format** parameter (see below), they expand the input key rather than the result value.

%[1-9] The patterns %1, %2, ... %9 are replaced by the corresponding most significant component of the input key's domain. If the input key is *user@mail.example.com*, then %1 is **com**, %2 is **example** and %3 is **mail**. If the input key is unqualified or does not have enough domain components to satisfy all the specified patterns, the query is suppressed and returns no results.

The **domain** parameter described below limits the input keys to addresses in matching domains. When the **domain** parameter is non-empty, SQL queries for unqualified addresses or addresses in non-matching domains are suppressed and return no results.

This parameter is available with Postfix 2.2. In prior releases the SQL query was built from the

separate parameters: **select_field**, **table**, **where_field** and **additional_conditions**. The mapping from the old parameters to the equivalent query is:

```
SELECT [select_field]
FROM [table]
WHERE [where_field] = '%s'
    [additional_conditions]
```

The '%s' in the **WHERE** clause expands to the escaped search string. With Postfix 2.2 these legacy parameters are used if the **query** parameter is not specified.

NOTE: DO NOT put quotes around the query parameter.

result_format (default: %s)

Format template applied to result attributes. Most commonly used to append (or prepend) text to the result. This parameter supports the following '%' expansions:

%% This is replaced by a literal '%' character.

%s This is replaced by the value of the result attribute. When result is empty it is skipped.

%u When the result attribute value is an address of the form user@domain, **%u** is replaced by the local part of the address. When the result has an empty localpart it is skipped.

%d When a result attribute value is an address of the form user@domain, **%d** is replaced by the domain part of the attribute value. When the result is unqualified it is skipped.

%[SUD1-9]

The upper-case and decimal digit expansions interpolate the parts of the input key rather than the result. Their behavior is identical to that described with **query**, and in fact because the input key is known in advance, queries whose key does not contain all the information specified in the result template are suppressed and return no results.

For example, using "result_format = smtp:[%s]" allows one to use a mailHost attribute as the basis of a transport(5) table. After applying the result format, multiple values are concatenated as comma separated strings. The expansion_limit and parameter explained below allows one to restrict the number of values in the result, which is especially useful for maps that must return at most one value.

The default value **%s** specifies that each result value should be used as is.

This parameter is available with Postfix 2.2 and later.

NOTE: DO NOT put quotes around the result format!

domain (default: no domain list)

This is a list of domain names, paths to files, or dictionaries. When specified, only fully qualified search keys with a *non-empty* localpart and a matching domain are eligible for lookup: 'user' lookups, bare domain lookups and "@domain" lookups are not performed. This can significantly reduce the query load on the MySQL server.

```
domain = postfix.org, hash:/etc/postfix/searchdomains
```

It is best not to use SQL to store the domains eligible for SQL lookups.

This parameter is available with Postfix 2.2 and later.

NOTE: DO NOT define this parameter for local(8) aliases, because the input keys are always unqualified.

expansion_limit (default: 0)
A limit on the total number of result elements returned (as a comma separated list) by a lookup against the map. A setting of zero disables the limit. Lookups fail with a temporary error if the limit is exceeded. Setting the limit to 1 ensures that lookups do not return multiple values.

The following parameters can be used to fill in a SELECT template statement of the form:

```
SELECT [select_field]
FROM [table]
WHERE [where_field] = '%s'
    [additional_conditions]
```

The specifier %s is replaced by the search string, and is escaped so if it contains single quotes or other odd characters, it will not cause a parse error, or worse, a security problem.

As of Postfix 2.2 this interface is obsolete, it is replaced by the more general **query** interface described above. If the **query** parameter is defined, the legacy parameters are ignored. Please migrate to the new interface as the legacy interface may be removed in a future release.

select_field
The SQL "select" parameter. Example:
select_field = forw_addr

table The SQL "select .. from" table name. Example:
table = mxaliases

where_field
The SQL "select .. where" parameter. Example:
where_field = alias

additional_conditions
Additional conditions to the SQL query. Example:
additional_conditions = AND status = 'paid'

SEE ALSO

postmap(1), Postfix lookup table maintenance
postconf(5), configuration parameters
ldap_table(5), LDAP lookup tables
pgsql_table(5), PostgreSQL lookup tables

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview
MYSQL_README, Postfix MYSQL client guide

LICENSE

The Secure Mailer license must be distributed with this software.

HISTORY

MySQL support was introduced with Postfix version 1.0.

AUTHOR(S)

Original implementation by:
Scott Cotton, Joshua Marcus
IC Group, Inc.

Further enhancements by:
Liviu Daia
Institute of Mathematics of the Romanian Academy
P.O. BOX 1-764

RO-014700 Bucharest, ROMANIA

NAME

named.conf − configuration file for named

SYNOPSIS

named.conf

DESCRIPTION

named.conf is the configuration file for **named**. Statements are enclosed in braces and terminated with a semi−colon. Clauses in the statements are also semi−colon terminated. The usual comment styles are supported:

C style: /* */

C++ style: // to end of line

Unix style: # to end of line

ACL

acl *string* { *address_match_element*; ... };

KEY

key *domain_name* {
    algorithm *string*;
    secret *string*;
};

MASTERS

masters *string* [port *integer*] {
    (*masters* | *ipv4_address* [port *integer*] |
    ipv6_address [port *integer*]) [key *string*]; ...
};

SERVER

server (*ipv4_address[/prefixlen]* | *ipv6_address[/prefixlen]*) {
    bogus *boolean*;
    edns *boolean*;
    edns−udp−size *integer*;
    max−udp−size *integer*;
    provide−ixfr *boolean*;
    request−ixfr *boolean*;
    keys *server_key*;
    transfers *integer*;
    transfer−format (many−answers | one−answer);
    transfer−source (*ipv4_address* | *)
        [port (*integer* | *)];
    transfer−source−v6 (*ipv6_address* | *)
        [port (*integer* | *)];
    support−ixfr *boolean*; // obsolete
};

TRUSTED−KEYS

trusted−keys {
    domain_name flags protocol algorithm key; ...
};

CONTROLS

controls {
    inet (*ipv4_address* | *ipv6_address* | *)
        [port (*integer* | *)]
        allow { *address_match_element*; ... }
        [keys { *string*; ... }];

```
        unix unsupported; // not implemented
};
```

LOGGING

```
logging {
        channel string {
                file log_file;
                syslog optional_facility;
                null;
                stderr;
                severity log_severity;
                print−time boolean;
                print−severity boolean;
                print−category boolean;
        };
        category string { string; ... };
};
```

LWRES

```
lwres {
        listen−on [ port integer ] {
                ( ipv4_address | ipv6_address ) [ port integer ]; ...
        };
        view string optional_class;
        search { string; ... };
        ndots integer;
};
```

OPTIONS

```
options {
        avoid−v4−udp−ports { port; ... };
        avoid−v6−udp−ports { port; ... };
        blackhole { address_match_element; ... };
        coresize size;
        datasize size;
        directory quoted_string;
        dump−file quoted_string;
        files size;
        heartbeat−interval integer;
        host−statistics boolean; // not implemented
        host−statistics−max number; // not implemented
        hostname ( quoted_string | none );
        interface−interval integer;
        listen−on [ port integer ] { address_match_element; ... };
        listen−on−v6 [ port integer ] { address_match_element; ... };
        match−mapped−addresses boolean;
        memstatistics−file quoted_string;
        pid−file ( quoted_string | none );
        port integer;
        querylog boolean;
        recursing−file quoted_string;
        random−device quoted_string;
        recursive−clients integer;
        serial−query−rate integer;
        server−id ( quoted_string | none |;
        stacksize size;
```

statistics−file *quoted_string*;
statistics−interval *integer*; // not yet implemented
tcp−clients *integer*;
tcp−listen−queue *integer*;
tkey−dhkey *quoted_string integer*;
tkey−gssapi−credential *quoted_string*;
tkey−domain *quoted_string*;
transfers−per−ns *integer*;
transfers−in *integer*;
transfers−out *integer*;
use−ixfr *boolean*;
version (*quoted_string* | none);
allow−recursion { *address_match_element*; ... };
sortlist { *address_match_element*; ... };
topology { *address_match_element*; ... }; // not implemented
auth−nxdomain *boolean*; // default changed
minimal−responses *boolean*;
recursion *boolean*;
rrset−order {
    [class *string*] [type *string*]
    [name *quoted_string*] *string string*; ...
};
provide−ixfr *boolean*;
request−ixfr *boolean*;
rfc2308−type1 *boolean*; // not yet implemented
additional−from−auth *boolean*;
additional−from−cache *boolean*;
query−source ((*ipv4_address* | *) | [address (*ipv4_address* | *)]) [port (*integer* | *)];
query−source−v6 ((*ipv6_address* | *) | [address (*ipv6_address* | *)]) [port (*integer* | *)];
cleaning−interval *integer*;
min−roots *integer*; // not implemented
lame−ttl *integer*;
max−ncache−ttl *integer*;
max−cache−ttl *integer*;
transfer−format (many−answers | one−answer);
max−cache−size *size_no_default*;
max−acache−size *size_no_default*;
clients−per−query *number*;
max−clients−per−query *number*;
check−names (master | slave | response)
    (fail | warn | ignore);
check−mx (fail | warn | ignore);
check−integrity *boolean*;
check−mx−cname (fail | warn | ignore);
check−srv−cname (fail | warn | ignore);
cache−file *quoted_string*; // test option
suppress−initial−notify *boolean*; // not yet implemented
preferred−glue *string*;
dual−stack−servers [port *integer*] {
    (*quoted_string* [port *integer*] |
    ipv4_address [port *integer*] |
    ipv6_address [port *integer*]); ...
};
edns−udp−size *integer*;

max−udp−size *integer*;
root−delegation−only [exclude { *quoted_string*; ... }];
disable−algorithms *string* { *string*; ... };
dnssec−enable *boolean*;
dnssec−validation *boolean*;
dnssec−lookaside *string* trust−anchor *string*;
dnssec−must−be−secure *string boolean*;
dnssec−accept−expired *boolean*;
empty−server *string*;
empty−contact *string*;
empty−zones−enable *boolean*;
disable−empty−zone *string*;
dialup *dialuptype*;
ixfr−from−differences *ixfrdiff*;
allow−query { *address_match_element*; ... };
allow−query−cache { *address_match_element*; ... };
allow−transfer { *address_match_element*; ... };
allow−update { *address_match_element*; ... };
allow−update−forwarding { *address_match_element*; ... };
update−check−ksk *boolean*;
notify *notifytype*;
notify−source (*ipv4_address* | *) [port (*integer* | *)];
notify−source−v6 (*ipv6_address* | *) [port (*integer* | *)];
notify−delay *seconds*;
also−notify [port *integer*] { (*ipv4_address* | *ipv6_address*)
[port *integer*]; ... };
allow−notify { *address_match_element*; ... };
forward (first | only);
forwarders [port *integer*] {
(*ipv4_address* | *ipv6_address*) [port *integer*]; ...
};
max−journal−size *size_no_default*;
max−transfer−time−in *integer*;
max−transfer−time−out *integer*;
max−transfer−idle−in *integer*;
max−transfer−idle−out *integer*;
max−retry−time *integer*;
min−retry−time *integer*;
max−refresh−time *integer*;
min−refresh−time *integer*;
multi−master *boolean*;
sig−validity−interval *integer*;
transfer−source (*ipv4_address* | *)
[port (*integer* | *)];
transfer−source−v6 (*ipv6_address* | *)
[port (*integer* | *)];
alt−transfer−source (*ipv4_address* | *)
[port (*integer* | *)];
alt−transfer−source−v6 (*ipv6_address* | *)
[port (*integer* | *)];
use−alt−transfer−source *boolean*;
zone−statistics *boolean*;
key−directory *quoted_string*;
zero−no−soa−ttl *boolean*;

```
        zero-no-soa-ttl-cache boolean;
        allow-v6-synthesis { address_match_element; ... }; // obsolete
        deallocate-on-exit boolean; // obsolete
        fake-iquery boolean; // obsolete
        fetch-glue boolean; // obsolete
        has-old-clients boolean; // obsolete
        maintain-ixfr-base boolean; // obsolete
        max-ixfr-log-size size; // obsolete
        multiple-cnames boolean; // obsolete
        named-xfer quoted_string; // obsolete
        serial-queries integer; // obsolete
        treat-cr-as-space boolean; // obsolete
        use-id-pool boolean; // obsolete
};
```

VIEW

```
view string optional_class {
        match-clients { address_match_element; ... };
        match-destinations { address_match_element; ... };
        match-recursive-only boolean;
        key string {
                algorithm string;
                secret string;
        };
        zone string optional_class {
                ...
        };
        server ( ipv4_address[/prefixlen] | ipv6_address[/prefixlen] ) {
                ...
        };
        trusted-keys {
                string integer integer integer quoted_string; ...
        };
        allow-recursion { address_match_element; ... };
        sortlist { address_match_element; ... };
        topology { address_match_element; ... }; // not implemented
        auth-nxdomain boolean; // default changed
        minimal-responses boolean;
        recursion boolean;
        rrset-order {
                [ class string ] [ type string ]
                [ name quoted_string ] string string; ...
        };
        provide-ixfr boolean;
        request-ixfr boolean;
        rfc2308-type1 boolean; // not yet implemented
        additional-from-auth boolean;
        additional-from-cache boolean;
        query-source ( ( ipv4_address | * ) | [ address ( ipv4_address | * ) ] ) [ port ( integer | * ) ];
        query-source-v6 ( ( ipv6_address | * ) | [ address ( ipv6_address | * ) ] ) [ port ( integer | * ) ];
        cleaning-interval integer;
        min-roots integer; // not implemented
        lame-ttl integer;
        max-ncache-ttl integer;
        max-cache-ttl integer;
```

transfer−format (many−answers | one−answer);
max−cache−size *size_no_default*;
max−acache−size *size_no_default*;
clients−per−query *number*;
max−clients−per−query *number*;
check−names (master | slave | response)
(fail | warn | ignore);
check−mx (fail | warn | ignore);
check−integrity *boolean*;
check−mx−cname (fail | warn | ignore);
check−srv−cname (fail | warn | ignore);
cache−file *quoted_string*; // test option
suppress−initial−notify *boolean*; // not yet implemented
preferred−glue *string*;
dual−stack−servers [port *integer*] {
(*quoted_string* [port *integer*] |
ipv4_address [port *integer*] |
ipv6_address [port *integer*]); ...
};
edns−udp−size *integer*;
max−udp−size *integer*;
root−delegation−only [exclude { *quoted_string*; ... }];
disable−algorithms *string* { *string*; ... };
dnssec−enable *boolean*;
dnssec−validation *boolean*;
dnssec−lookaside *string* trust−anchor *string*;
dnssec−must−be−secure *string boolean*;
dnssec−accept−expired *boolean*;
empty−server *string*;
empty−contact *string*;
empty−zones−enable *boolean*;
disable−empty−zone *string*;
dialup *dialuptype*;
ixfr−from−differences *ixfrdiff*;
allow−query { *address_match_element*; ... };
allow−query−cache { *address_match_element*; ... };
allow−transfer { *address_match_element*; ... };
allow−update { *address_match_element*; ... };
allow−update−forwarding { *address_match_element*; ... };
update−check−ksk *boolean*;
notify *notifytype*;
notify−source (*ipv4_address* | *) [port (*integer* | *)];
notify−source−v6 (*ipv6_address* | *) [port (*integer* | *)];
notify−delay *seconds*;
also−notify [port *integer*] { (*ipv4_address* | *ipv6_address*)
[port *integer*]; ... };
allow−notify { *address_match_element*; ... };
forward (first | only);
forwarders [port *integer*] {
(*ipv4_address* | *ipv6_address*) [port *integer*]; ...
};
max−journal−size *size_no_default*;
max−transfer−time−in *integer*;
max−transfer−time−out *integer*;

```
                max-transfer-idle-in integer;
                max-transfer-idle-out integer;
                max-retry-time integer;
                min-retry-time integer;
                max-refresh-time integer;
                min-refresh-time integer;
                multi-master boolean;
                sig-validity-interval integer;
                transfer-source ( ipv4_address | * )
                        [ port ( integer | * ) ];
                transfer-source-v6 ( ipv6_address | * )
                        [ port ( integer | * ) ];
                alt-transfer-source ( ipv4_address | * )
                        [ port ( integer | * ) ];
                alt-transfer-source-v6 ( ipv6_address | * )
                        [ port ( integer | * ) ];
                use-alt-transfer-source boolean;
                zone-statistics boolean;
                key-directory quoted_string;
                zero-no-soa-ttl boolean;
                zero-no-soa-ttl-cache boolean;
                allow-v6-synthesis { address_match_element; ... }; // obsolete
                fetch-glue boolean; // obsolete
                maintain-ixfr-base boolean; // obsolete
                max-ixfr-log-size size; // obsolete
        };
```

ZONE

```
        zone string optional_class {
                type ( master | slave | stub | hint |
                        forward | delegation-only );
                file quoted_string;
                masters [ port integer ] {
                        ( masters |
                        ipv4_address [port integer] |
                        ipv6_address [ port integer ] ) [ key string ]; ...
                };
                database string;
                delegation-only boolean;
                check-names ( fail | warn | ignore );
                check-mx ( fail | warn | ignore );
                check-integrity boolean;
                check-mx-cname ( fail | warn | ignore );
                check-srv-cname ( fail | warn | ignore );
                dialup dialuptype;
                ixfr-from-differences boolean;
                journal quoted_string;
                zero-no-soa-ttl boolean;
                allow-query { address_match_element; ... };
                allow-transfer { address_match_element; ... };
                allow-update { address_match_element; ... };
                allow-update-forwarding { address_match_element; ... };
                update-policy {
                        ( grant | deny ) string
                        ( name | subdomain | wildcard | self ) string
```

```
                *rrtypelist*; ...
        };
        update−check−ksk boolean;
        notify notifytype;
        notify−source ( ipv4_address | * ) [ port ( integer | * ) ];
        notify−source−v6 ( ipv6_address | * ) [ port ( integer | * ) ];
        notify−delay seconds;
        also−notify [ port integer ] { ( ipv4_address | ipv6_address )
                [ port integer ]; ... };
        allow−notify { address_match_element; ... };
        forward ( first | only );
        forwarders [ port integer ] {
                ( ipv4_address | ipv6_address ) [ port integer ]; ...
        };
        max−journal−size size_no_default;
        max−transfer−time−in integer;
        max−transfer−time−out integer;
        max−transfer−idle−in integer;
        max−transfer−idle−out integer;
        max−retry−time integer;
        min−retry−time integer;
        max−refresh−time integer;
        min−refresh−time integer;
        multi−master boolean;
        sig−validity−interval integer;
        transfer−source ( ipv4_address | * )
                [ port ( integer | * ) ];
        transfer−source−v6 ( ipv6_address | * )
                [ port ( integer | * ) ];
        alt−transfer−source ( ipv4_address | * )
                [ port ( integer | * ) ];
        alt−transfer−source−v6 ( ipv6_address | * )
                [ port ( integer | * ) ];
        use−alt−transfer−source boolean;
        zone−statistics boolean;
        key−directory quoted_string;
        ixfr−base quoted_string; // obsolete
        ixfr−tmp−file quoted_string; // obsolete
        maintain−ixfr−base boolean; // obsolete
        max−ixfr−log−size size; // obsolete
        pubkey integer integer integer quoted_string; // obsolete
};
```

FILES

/etc/named.conf

SEE ALSO

named(8), **rndc**(8), **BIND 9 Administrator Reference Manual**().

COPYRIGHT

NAME

netconfig — network configuration data base

SYNOPSIS

/etc/netconfig

DESCRIPTION

The **netconfig** file defines a list of "transport names", describing their semantics and protocol. In NetBSD, this file is only used by the RPC library code.

Entries have the following format:

```
network_id semantics flags family protoname device libraries
```

Entries consist of the following fields:

network_id The name of the transport described.

semantics Describes the semantics of the transport. This can be one of:

- **tpi_clts** Connectionless transport.
- **tpi_cots** Connection-oriented transport
- **tpi_cots_ord** Connection-oriented, ordered transport.
- **tpi_raw** A raw connection.

flags This field is either blank (specified by "-"), or contains a "v", meaning visible to the `getnetconfig`(3) function.

family The protocol family of the transport. This is currently one of:

- **inet6** The IPv6 (`PF_INET6`) family of protocols.
- **inet** The IPv4 (`PF_INET`) family of protocols.
- **loopback** The `PF_LOCAL` protocol family.

protoname The name of the protocol used for this transport. Can currently be either **udp**, **tcp** or empty.

device This field is always empty in NetBSD.

libraries This field is always empty in NetBSD.

The order of entries in this file will determine which transport will be preferred by the RPC library code, given a match on a specified network type. For example, if a sample network config file would look like this:

```
udp6       tpi_clts      v     inet6    udp     -       -
tcp6       tpi_cots_ord  v     inet6    tcp     -       -
udp        tpi_clts      v     inet     udp     -       -
tcp        tpi_cots_ord  v     inet     tcp     -       -
rawip      tpi_raw       -     inet      -      -       -
local      tpi_cots_ord  -     loopback  -      -       -
```

then using the network type **udp** in calls to the RPC library function (see `rpc`(3)) will make the code first try **udp6**, and then **udp**.

`getnetconfig`(3) and associated functions will parse this file and return structures of the following format:

```
struct netconfig {
    char *nc_netid;              /* Network ID */
```

```
        unsigned long nc_semantics;  /* Semantics (see below) */
        unsigned long nc_flag;       /* Flags (see below) */
        char *nc_protofmly;          /* Protocol family */
        char *nc_proto;              /* Protocol name */
        char *nc_device;             /* Network device pathname (unused) */
        unsigned long nc_nlookups;   /* Number of lookup libs (unused) */
        char **nc_lookups;           /* Names of the libraries (unused) */
        unsigned long nc_unused[9];  /* reserved */
    };
```

FILES

/etc/netconfig

SEE ALSO

getnetconfig(3), getnetpath(3)

NAME

netgroup — defines network groups

SYNOPSIS

netgroup

DESCRIPTION

The **netgroup** file specifies "netgroups", which are sets of **(host, user, domain)** tuples that are to be given similar network access.

Each line in the file consists of a netgroup name followed by a list of the members of the netgroup. Each member can be either the name of another netgroup or a specification of a tuple as follows:

```
(host, user, domain)
```

where the **host**, **user**, and **domain** are character string names for the corresponding component. Any of the comma separated fields may be empty to specify a "wildcard" value or may consist of the string "`-`" to specify "no valid value". The members of the list may be separated by whitespace; the "\" character may be used at the end of a line to specify line continuation. The functions specified in `getnetgrent`(3) should normally be used to access the **netgroup** database.

If 'files' is specified for the 'netgroup' database in `nsswitch.conf`(5), (or no 'netgroup' entry is specified), then these functions operate on the `db`(3) version of the **netgroup (netgroup.db)** file which can be generated using `netgroup_mkdb`(8). If 'nis' is specified then the NIS maps 'netgroup', 'netgroup.byhost', and 'netgroup.byuser' are used.

Lines that begin with a # are treated as comments.

FILES

`/etc/netgroup.db` the netgroup database.

SEE ALSO

`getnetgrent`(3), `exports`(5), `nsswitch.conf`(5), `netgroup_mkdb`(8)

COMPATIBILITY

The file format is compatible with that of various vendors, however it appears that not all vendors use an identical format.

BUGS

The interpretation of access restrictions based on the member tuples of a netgroup is left up to the various network applications.

NAME

netid — NIS network credential file

DESCRIPTION

Files in **netid** format are rare. One lives in the NIS map `netid.byname`. The format is rather simple. Each row consists of two items, a key and a value. When created by `mknetid`(8) there are three kind of records.

The first type is information about which gids a uid has:

```
unix.<uid>@<yp-domain> <uid>:<gid>,<gid>
```

The second type has informations about hosts:

```
unix.<hostname>@<yp-domain> 0:<hostname>
```

The third type is records from a **netid** file other than the two types above.

FILES

`/etc/netid` A file for lines not generated automatically by `mknetid`(8).

EXAMPLES

A configuration file might appear as follows:

```
unix.10714@kaka 10714:400,10
unix.jodie@kaka 0:jodie
```

SEE ALSO

`mknetid`(8), `nis`(8)

AUTHORS

Mats O Jansson ⟨moj@stacken.kth.se⟩

NAME

networks — Internet Protocol network name data base

DESCRIPTION

The **networks** file is used as a local source to translate between Internet Protocol (IP) network addresses and network names (and vice versa). It can be used in conjunction with the DNS, as controlled by nsswitch.conf(5).

While the **networks** file was originally intended to be an exhaustive list of all IP networks that the local host could communicate with, distribution and update of such a list for the world-wide Internet (or, indeed, for any large "enterprise" network) has proven to be prohibitive, so the Domain Name System (DNS) is used instead, except as noted.

For each IP network a single line should be present with the following information:

```
name network [alias ...]
```

These are:

name	Official network name
network	IP network number
alias	Network alias

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Network number may be specified in the conventional dot (".") notation using the inet_network(3) routine from the IP address manipulation library, inet(3). Network names may contain "a" through "z", zero through nine, and dash.

IP network numbers on the Internet are generally assigned to a site by its Internet Service Provider (ISP), who, in turn, get network address space assigned to them by one of the regional Internet Registries (e.g. ARIN, RIPE NCC, APNIC). These registries, in turn, answer to the Internet Assigned Numbers Authority (IANA).

If a site changes its ISP from one to another, it will generally be required to change all its assigned IP addresses as part of the conversion; that is, return the previous network numbers to the previous ISP, and assign addresses to its hosts from IP network address space given by the new ISP. Thus, it is best for a savvy network manager to configure his hosts for easy renumbering, to preserve his ability to easily change his ISP should the need arise.

FILES

/etc/networks The **networks** file resides in /etc.

SEE ALSO

getnetent(3), nsswitch.conf(5), resolv.conf(5), hostname(7), dhclient(8), dhcpd(8), named(8)

Classless IN-ADDR.ARPA delegation, RFC 2317, March 1998.

Address Allocation for Private Internets, RFC 1918, February 1996.

Network 10 Considered Harmful, RFC 1627, July 1994.

Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy, RFC 1519, September 1993.

DNS Encoding of Network Names and Other Types, RFC 1101, April 1989.

HISTORY

The **networks** file format appeared in 4.2BSD.

NAME

nisplus_table − Postfix NIS+ client

SYNOPSIS

postmap -q "*string***" "nisplus:[***name***=%s];***name.name.***"**

postmap -q - "nisplus:[*name***=%s];***name.name.***"** <*inputfile*

DESCRIPTION

The Postfix mail system uses optional lookup tables. These tables are usually in **dbm** or **db** format. Alternatively, lookup tables can be specified as NIS+ databases.

To find out what types of lookup tables your Postfix system supports use the "**postconf -m**" command.

To test Postfix NIS+ lookup tables, use the "**postmap -q**" command as described in the SYNOPSIS above.

QUERY SYNTAX

Most of the NIS+ query is specified via the NIS+ map name. The general format of a Postfix NIS+ map name is as follows:

nisplus:[*name***=%s];***name.name.name***.:***column*

Postfix NIS+ map names differ from what one normally would use with commands such as **niscat**:

- With each NIS+ table lookup, "**%s**" is replaced by a version of the lookup string. There can be only one "**%s**" instance in a Postfix NIS+ map name.
- Postfix NIS+ map names use "**;**" instead of "**,**", because the latter character is special in the Postfix main.cf file. Postfix replaces "**;**" characters in the map name by "**,**" before making NIS+ queries.
- The ":*column*" part in the NIS+ map name is not part of the actual NIS+ query. Instead, it specifies the number of the table column that provides the lookup result. When no ":*column*" is specified the first column (1) is used.

EXAMPLE

A NIS+ aliases map might be queried as follows:

```
alias_maps = dbm:/etc/mail/aliases,
    nisplus:[alias=%s];mail_aliases.org_dir.$mydomain.:1
```

This queries the local aliases file before the NIS+ file.

SEE ALSO

postmap(1), Postfix lookup table manager

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Geoff Gibbs
UK-HGMP-RC
Hinxton
Cambridge
CB10 1SB, UK

Adopted and adapted by:
Wietse Venema

IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

nologin — file disallowing and containing reason for disallowing logins

DESCRIPTION

The file /etc/nologin, if it exists, causes the login procedure, used by programs such as login(1), to terminate. The program may display the contents of /etc/nologin to the user before exiting.

This file is a simple mechanism to temporarily prevent incoming logins. As such, the file /etc/nologin is created by shutdown(8) five minutes before system shutdown, or immediately if shutdown is in less than five minutes. The file /etc/nologin is removed just before shutdown(8) exits.

To disable logins on a per-account basis, see nologin(8).

The file /etc/nologin has no affect on the login procedure for the root user.

FILES

/etc/nologin The **nologin** file resides in /etc.

EXAMPLES

```
NO LOGINS: System going down at 18:22
```

SEE ALSO

login(1), ftpd(8), nologin(8), rshd(8), shutdown(8), sshd(8)

NAME

nsswitch.conf — name-service switch configuration file

DESCRIPTION

The **nsswitch.conf** file specifies how the nsdispatch(3) (name-service switch dispatcher) routines in the C library should operate.

The configuration file controls how a process looks up various databases containing information regarding hosts, users (passwords), groups, netgroups, etc. Each database comes from a source (such as local files, DNS, and NIS), and the order to look up the sources is specified in **nsswitch.conf**.

Each entry in **nsswitch.conf** consists of a database name, and a space separated list of sources. Each source can have an optional trailing criterion that determines whether the next listed source is used, or the search terminates at the current source. Each criterion consists of one or more status codes, and actions to take if that status code occurs.

Sources

The following sources are implemented:

Source	Description
files	Local files, such as /etc/hosts, and /etc/passwd.
dns	Internet Domain Name System. “hosts” and ‘networks’ use **IN** class entries, all other databases use **HS** class (Hesiod) entries.
nis	NIS (formerly YP)
compat	support ‘+/-’ in the “passwd” and “group” databases. If this is present, it must be the only source for that entry.

Databases

The following databases are used by the following C library functions:

Database	Used by
group	getgrent(3)
hosts	gethostbyname(3)
netgroup	getnetgrent(3)
networks	getnetbyname(3)
passwd	getpwent(3)
shells	getusershell(3)

Status codes

The following status codes are available:

Status	Description
success	The requested entry was found.
notfound	The entry is not present at this source.
tryagain	The source is busy, and may respond to retries.
unavail	The source is not responding, or entry is corrupt.

Actions

For each of the status codes, one of two actions is possible:

Action	Description
continue	Try the next source
return	Return with the current result

Format of file

A BNF description of the syntax of **nsswitch.conf** is:

```
<entry>     ::= <database> ":" [<source> [<criteria>]]*
<criteria>  ::= "[" <criterion>+ "]"
<criterion> ::= <status> "=" <action>
<status>    ::= "success" | "notfound" | "unavail" | "tryagain"
<action>    ::= "return" | "continue"
```

Each entry starts on a new line in the file. A '#' delimits a comment to end of line. Blank lines are ignored. A '\' at the end of a line escapes the newline, and causes the next line to be a continuation of the current line. All entries are case-insensitive.

The default criteria is to return on "success", and continue on anything else (i.e, `[success=return notfound=continue unavail=continue tryagain=continue]`).

Compat mode: +/- syntax

In historical multi-source implementations, the '+' and '-' characters are used to specify the importing of user password and group information from NIS. Although **nsswitch.conf** provides alternative methods of accessing distributed sources such as NIS, specifying a sole source of "compat" will provide the historical behaviour.

An alternative source for the information accessed via '+/-' can be used by specifying "passwd_compat: source". "source" in this case can be 'dns', 'nis', or any other source except for 'files' and 'compat'.

Notes

Historically, many of the databases had enumeration functions, often of the form **getXXXent**(). These made sense when the databases were in local files, but don't make sense or have lesser relevance when there are possibly multiple sources, each of an unknown size. The interfaces are still provided for compatibility, but the source may not be able to provide complete entries, or duplicate entries may be retrieved if multiple sources that contain similar information are specified.

To ensure compatibility with previous and current implementations, the "compat" source must appear alone for a given database.

Default source lists

If, for any reason, **nsswitch.conf** doesn't exist, or it has missing or corrupt entries, `nsdispatch`(3) will default to an entry of "files" for the requested database. Exceptions are:

Database	Default source list
group	compat
group_compat	nis
hosts	files dns
netgroup	files [notfound=return] nis
passwd	compat
passwd_compat	nis

FILES

`/etc/nsswitch.conf` The file **nsswitch.conf** resides in `/etc`.

EXAMPLES

To lookup hosts in `/etc/hosts` and then from the DNS, and lookup user information from NIS then files, use:

```
hosts:  files dns
passwd: nis [notfound=return] files
group:  nis [notfound=return] files
```

The criteria “[notfound=return]” sets a policy of "if the user is notfound in nis, don’t try files." This treats nis as the authoritative source of information, except when the server is down.

SEE ALSO

`getent`(1), `nsdispatch`(3), `resolv.conf`(5), `named`(8), `ypbind`(8)

HISTORY

The **`nsswitch.conf`** file format first appeared in NetBSD 1.4.

AUTHORS

Luke Mewburn ⟨lukem@NetBSD.org⟩ wrote this freely distributable name-service switch implementation, using ideas from the ULTRIX `svc.conf`(5) and Solaris `nsswitch.conf`(4) manual pages.

NAME

config − OpenSSL CONF library configuration files

DESCRIPTION

The OpenSSL CONF library can be used to read configuration files. It is used for the OpenSSL master configuration file **openssl.cnf** and in a few other places like **SPKAC** files and certificate extension files for the **x509** utility. OpenSSL applications can also use the CONF library for their own purposes.

A configuration file is divided into a number of sections. Each section starts with a line **[section_name]** and ends when a new section is started or end of file is reached. A section name can consist of alphanumeric characters and underscores.

The first section of a configuration file is special and is referred to as the **default** section this is usually unnamed and is from the start of file until the first named section. When a name is being looked up it is first looked up in a named section (if any) and then the default section.

The environment is mapped onto a section called **ENV**.

Comments can be included by preceding them with the **#** character

Each section in a configuration file consists of a number of name and value pairs of the form **name=value**

The **name** string can contain any alphanumeric characters as well as a few punctuation symbols such as **. , ;** and **_**.

The **value** string consists of the string following the **=** character until end of line with any leading and trailing white space removed.

The value string undergoes variable expansion. This can be done by including the form **$var** or **${var}**: this will substitute the value of the named variable in the current section. It is also possible to substitute a value from another section using the syntax **$section::name** or **${section::name}**. By using the form **$ENV::name** environment variables can be substituted. It is also possible to assign values to environment variables by using the name **ENV::name**, this will work if the program looks up environment variables using the **CONF** library instead of calling ***getenv()*** directly.

It is possible to escape certain characters by using any kind of quote or the \ character. By making the last character of a line a \ a **value** string can be spread across multiple lines. In addition the sequences **\n**, **\r**, **\b** and **\t** are recognized.

OPENSSL LIBRARY CONFIGURATION

In OpenSSL 0.9.7 and later applications can automatically configure certain aspects of OpenSSL using the master OpenSSL configuration file, or optionally an alternative configuration file. The **openssl** utility includes this functionality: any sub command uses the master OpenSSL configuration file unless an option is used in the sub command to use an alternative configuration file.

To enable library configuration the default section needs to contain an appropriate line which points to the main configuration section. The default name is **openssl_conf** which is used by the **openssl** utility. Other applications may use an alternative name such as **myapplicaton_conf**.

The configuration section should consist of a set of name value pairs which contain specific module configuration information. The **name** represents the name of the *configuration module* the meaning of the **value** is module specific: it may, for example, represent a further configuration section containing configuration module specific information. E.g.

```
openssl_conf = openssl_init

[openssl_init]

oid_section = new_oids
engines = engine_section

[new_oids]

... new oids here ...
```

```
[engine_section]

... engine stuff here ...
```

Currently there are two configuration modules. One for ASN1 objects another for ENGINE configuration.

ASN1 OBJECT CONFIGURATION MODULE

This module has the name **oid_section**. The value of this variable points to a section containing name value pairs of OIDs: the name is the OID short and long name, the value is the numerical form of the OID. Although some of the **openssl** utility sub commands already have their own ASN1 OBJECT section functionality not all do. By using the ASN1 OBJECT configuration module **all** the **openssl** utility sub commands can see the new objects as well as any compliant applications. For example:

```
[new_oids]

some_new_oid = 1.2.3.4
some_other_oid = 1.2.3.5
```

In OpenSSL 0.9.8 it is also possible to set the value to the long name followed by a comma and the numerical OID form. For example:

```
shortName = some object long name, 1.2.3.4
```

ENGINE CONFIGURATION MODULE

This ENGINE configuration module has the name **engines**. The value of this variable points to a section containing further ENGINE configuration information.

The section pointed to by **engines** is a table of engine names (though see **engine_id** below) and further sections containing configuration informations specific to each ENGINE.

Each ENGINE specific section is used to set default algorithms, load dynamic, perform initialization and send ctrls. The actual operation performed depends on the *command* name which is the name of the name value pair. The currently supported commands are listed below.

For example:

```
[engine_section]

# Configure ENGINE named "foo"
foo = foo_section
# Configure ENGINE named "bar"
bar = bar_section

[foo_section]
... foo ENGINE specific commands ...

[bar_section]
... "bar" ENGINE specific commands ...
```

The command **engine_id** is used to give the ENGINE name. If used this command must be first. For example:

```
[engine_section]
# This would normally handle an ENGINE named "foo"
foo = foo_section

[foo_section]
# Override default name and use "myfoo" instead.
engine_id = myfoo
```

The command **dynamic_path** loads and adds an ENGINE from the given path. It is equivalent to sending the ctrls **SO_PATH** with the path argument followed by **LIST_ADD** with value 2 and **LOAD** to the dynamic ENGINE. If this is not the required behaviour then alternative ctrls can be sent directly to the dynamic ENGINE using ctrl commands.

The command **init** determines whether to initialize the ENGINE. If the value is **0** the ENGINE will not be initialized, if **1** and attempt it made to initialized the ENGINE immediately. If the **init** command is not present then an attempt will be made to initialize the ENGINE after all commands in its section have been processed.

The command **default_algorithms** sets the default algorithms an ENGINE will supply using the functions ***ENGINE_set_default_string()***

If the name matches none of the above command names it is assumed to be a ctrl command which is sent to the ENGINE. The value of the command is the argument to the ctrl command. If the value is the string **EMPTY** then no value is sent to the command.

For example:

```
[engine_section]

# Configure ENGINE named "foo"
foo = foo_section

[foo_section]
# Load engine from DSO
dynamic_path = /some/path/fooengine.so
# A foo specific ctrl.
some_ctrl = some_value
# Another ctrl that doesn't take a value.
other_ctrl = EMPTY
# Supply all default algorithms
default_algorithms = ALL
```

NOTES

If a configuration file attempts to expand a variable that doesn't exist then an error is flagged and the file will not load. This can happen if an attempt is made to expand an environment variable that doesn't exist. For example in a previous version of OpenSSL the default OpenSSL master configuration file used the value of **HOME** which may not be defined on non Unix systems and would cause an error.

This can be worked around by including a **default** section to provide a default value: then if the environment lookup fails the default value will be used instead. For this to work properly the default value must be defined earlier in the configuration file than the expansion. See the **EXAMPLES** section for an example of how to do this.

If the same variable exists in the same section then all but the last value will be silently ignored. In certain circumstances such as with DNs the same field may occur multiple times. This is usually worked around by ignoring any characters before an initial **.** e.g.

```
1.OU="My first OU"
2.OU="My Second OU"
```

EXAMPLES

Here is a sample configuration file using some of the features mentioned above.

```
# This is the default section.

HOME=/temp
RANDFILE= ${ENV::HOME}/.rnd
configdir=$ENV::HOME/config

[ section_one ]

# We are now in section one.

# Quotes permit leading and trailing whitespace
any = " any variable name "
```

```
other = A string that can \
cover several lines \
by including \\ characters

message = Hello World\n

[ section_two ]

greeting = $section_one::message
```

This next example shows how to expand environment variables safely.

Suppose you want a variable called **tmpfile** to refer to a temporary filename. The directory it is placed in can determined by the the **TEMP** or **TMP** environment variables but they may not be set to any value at all. If you just include the environment variable names and the variable doesn't exist then this will cause an error when an attempt is made to load the configuration file. By making use of the default section both values can be looked up with **TEMP** taking priority and /**tmp** used if neither is defined:

```
TMP=/tmp
# The above value is used if TMP isn't in the environment
TEMP=$ENV::TMP
# The above value is used if TEMP isn't in the environment
tmpfile=${ENV::TEMP}/tmp.filename
```

BUGS

Currently there is no way to include characters using the octal **\nnn** form. Strings are all null terminated so nulls cannot form part of the value.

The escaping isn't quite right: if you want to use sequences like **\n** you can't use any quote escaping on the same line.

Files are loaded in a single pass. This means that an variable expansion will only work if the variables referenced are defined earlier in the file.

SEE ALSO

openssl_x509 (1), *openssl_req* (1), *openssl_ca* (1)

NAME

pam.conf — PAM policy file format

DESCRIPTION

The PAM library searches for policies in the following files, in decreasing order of preference:

1. /etc/pam.d/*service-name*
2. /etc/pam.conf
3. /usr/local/etc/pam.d/*service-name*
4. /usr/local/etc/pam.conf

If none of these locations contains a policy for the given service, the default policy is used instead, if it exists.

Entries in per-service policy files must be of one of the two forms below:

function-class control-flag module-path [*arguments ...*]
function-class **include** *other-service-name*

Entries in pam.conf-style policy files are of the same form, but are prefixed by an additional field specifying the name of the service they apply to.

In both types of policy files, blank lines are ignored, as is anything to the right of a '#' sign.

The *function-class* field specifies the class of functions the entry applies to, and is one of:

auth Authentication functions (pam_authenticate(3), pam_setcred(3))

account Account management functions (pam_acct_mgmt(3))

session Session handling functions (pam_open_session(3), pam_close_session(3))

password Password management functions (pam_chauthtok(3))

The *control-flag* field determines how the result returned by the module affects the flow of control through (and the final result of) the rest of the chain, and is one of:

required If this module succeeds, the result of the chain will be success unless a later module fails. If it fails, the rest of the chain still runs, but the final result will be failure regardless of the success of later modules.

requisite If this module succeeds, the result of the chain will be success unless a later module fails. If it module fails, the chain is broken and the result is failure.

sufficient If this module succeeds, the chain is broken and the result is success. If it fails, the rest of the chain still runs, but the final result will be failure unless a later module succeeds.

binding If this module succeeds, the chain is broken and the result is success. If it fails, the rest of the chain still runs, but the final result will be failure regardless of the success of later modules.

optional If this module succeeds, the result of the chain will be success unless a later module fails. If this module fails, the result of the chain will be failure unless a later module succeeds.

There are two exceptions to the above: **sufficient** and **binding** modules are treated as **optional** by pam_setcred(3), and in the PAM_PRELIM_CHECK phase of pam_chauthtok(3).

The *module-path* field specifies the name, or optionally the full path, of the module to call.

The remaining fields are passed as arguments to the module if and when it is invoked.

The **include** form of entry causes entries from a different chain (specified by *other-system-name*) to be included in the current one. This allows one to define system-wide policies which are then included into service-specific policies. The system-wide policy can then be modified without having to also modify each and every service-specific policy.

SEE ALSO

pam(3)

STANDARDS

X/Open Single Sign-On Service (XSSO) - Pluggable Authentication Modules, June 1997.

AUTHORS

The OpenPAM library was developed for the FreeBSD Project by ThinkSec AS and Network Associates Laboratories, the Security Research Division of Network Associates, Inc. under DARPA/SPAWAR contract N66001-01-C-8035 ("CBOSS"), as part of the DARPA CHATS research program.

This manual page was written by Dag-Erling Smørgrav ⟨des@FreeBSD.org⟩.

NAME

pam.conf — Pluggable Authentication Modules configuration file

DESCRIPTION

The **pam.conf** file specifies how Pluggable Authentication Modules (PAM) should operate. For an overview of the Pluggable Authentication Modules framework, see pam(8).

PAM may be configured using a single /etc/pam.conf configuration file or by using multiple configuration files, one for each PAM-aware service, located in the /etc/pam.d/ directory. If /etc/pam.d/ exists, /etc/pam.conf will be ignored. /etc/pam.d/ is the preferred method for configuring PAM.

PAM's configuration is based on "stacking" different modules together to form a processing chain for the task. A standard PAM configuration stanza is structured as follows:

[*service-name*] *module-type control-flag module-name* [*options*]

service-name is used only (and is mandatory) in /etc/pam.conf. It specifies the PAM-aware service whose PAM behavior is being configured. When /etc/pam.d/ is used, the name of the configuration file specifies the service.

module-type specifies which of the four classes of PAM module functionality is being configured. These four classes are *account* (account management), *auth* (authentication), *password* (password management), and *session* (session management).

control-flag specifies the behavior of the processing chain upon success or failure of the PAM module's authentication task. The following are valid values for *control-flag*:

binding — If the module succeeds and no earlier module in the chain has failed, the chain is immediately terminated and the request is granted. If the module fails, the rest of the chain is executed, but the request is ultimately denied.

requisite — If the module returns success, continue to execute the processing chain. If the module fails, immediately return the error code from the first 'required' failure.

required — If the module returns success, continue to execute the processing chain. If the module fails, record as a 'required' failure and continue to execute the processing chain. If there are any 'required' failures in the processing chain, the chain will ultimately return failure.

optional — If the module returns success, continue to execute the processing chain. If the module fails, record as an 'optional' failure and continue to execute the processing chain.

sufficient — If the module returns success and there have been no recorded 'required' failures, immediately return success without calling any subsequent modules in the processing chain. If the module fails, return as an 'optional' failure and continue to execute the processing chain.

module-name specifies the module to execute for this stanza. This is either an absolute path name or a path name relative to the default module location: /usr/lib/security.

options are additional options that may be specified for the module. Refer to the individual modules' documentation for more information on available options.

In addition to the standard configuration stanza format, there is an additional stanza format available when /etc/pam.d/ is used:

module-type include *service-name*

This stanza format provides a simple inheritance model for processing chains.

FILES

`/etc/pam.conf` monolithic PAM configuration file
`/etc/pam.d/` PAM service configuration file directory

EXAMPLES

The following *auth* processing chain for the "login" service (located in `/etc/pam.d/login`) performs the following tasks: allows the login if the old user and new user are the same, verifies that logins are not disabled using the `/var/run/nologin` file, allows Kerberos 5 password authentication, and requires standard UNIX password authentication if Kerberos 5 failed:

```
auth    sufficient      pam_self.so
auth    required        pam_nologin.so
auth    sufficient      pam_krb5.so
auth    required        pam_unix.so
```

NOTES

It is important to note that loading a chain will fail if any of the components of the chain fail to load or are not available. A common situation when this can happen is on a system that where components such as `kerberos`(1) or `crypto`(3) have not been installed. In that situation `pam_krb5`(8), `pam_ksu`(8), or `pam_ssh`(8) might not be present in the system. In order for a chain to load properly all non-present components must be removed from the chain.

SEE ALSO

`login`(1), `passwd`(1), `su`(1), `pam`(3), `pam`(8)

HISTORY

The **`pam.conf`** file format first appeared in NetBSD 3.0.

NAME

passwd, **master.passwd** — format of the password file

DESCRIPTION

The **passwd** files are the local source of password information. They can be used in conjunction with the Hesiod domain 'passwd' and the NIS maps 'passwd.byname', 'passwd.byuid', 'master.passwd.byname', and 'master.passwd.byuid', as controlled by nsswitch.conf(5).

The **master.passwd** file is readable only by root, and consists of newline separated ASCII records, one per user, containing ten colon (":") separated fields.

Each line has the form:

```
name:password:uid:gid:class:change:expire:gecos:home_dir:shell
```

These fields are as follows:

name	User's login name.
password	User's *encrypted* password.
uid	User's id.
gid	User's login group id.
class	User's login class.
change	Password change time.
expire	Account expiration time.
gecos	General information about the user.
home_dir	User's home directory.
shell	User's login shell.

Be aware that each line is limited to 1024 characters; longer ones will be ignored. This limit can be queried through sysconf(3) by using the _SC_GETPW_R_SIZE_MAX parameter.

The **passwd** file is generated from the **master.passwd** file by pwd_mkdb(8), has the *class*, *change*, and *expire* fields removed, and the *password* field replaced by a "*".

The *name* field is the login used to access the computer account, and the *uid* field is the number associated with it. They should both be unique across the system (and often across a group of systems) since they control file access.

While it is possible to have multiple entries with identical login names and/or identical user id's, it is usually a mistake to do so. Routines that manipulate these files will often return only one of the multiple entries, and that one by random selection.

The login name must never begin with a hyphen ("-"); also, it is strongly suggested that neither upper-case characters nor dots (".") be part of the name, as this tends to confuse mailers. No field may contain a colon (":") as this has been used historically to separate the fields in the user database.

The *password* field is the *encrypted* form of the password. If the *password* field is empty, no password will be required to gain access to the machine. This is almost invariably a mistake. Because these files contain the encrypted user passwords, they should not be readable by anyone without appropriate privileges. For the possible ciphers used in this field see passwd.conf(5).

The *gid* field is the group that the user will be placed in upon login. Since this system supports multiple groups (see groups(1)) this field currently has little special meaning.

The *class* field is a key for a user's login class. Login classes are defined in login.conf(5), which is a termcap(5) style database of user attributes, accounting, resource and environment settings.

The *change* field is the number of seconds from the epoch, UTC, until the password for the account must be changed. This field may be left empty to turn off the password aging feature. If this is set to "-1" then the user will be prompted to change their password at the next login.

The *expire* field is the number of seconds from the epoch, UTC, until the account expires. This field may be left empty to turn off the account aging feature.

If either of the *change* or *expire* fields are set, the system will remind the user of the impending change or expiry if they login within a configurable period (defaulting to 14 days) before the event.

The *gecos* field normally contains comma (",") separated subfields as follows:

name	user's full name
office	user's office number
wphone	user's work phone number
hphone	user's home phone number

The full name may contain an ampersand ("&") which will be replaced by the capitalized login name when the gecos field is displayed or used by various programs such as finger(1), sendmail(8), etc.

The office and phone number subfields are used by the finger(1) program, and possibly other applications.

The user's home directory is the full UNIX path name where the user will be placed on login.

The shell field is the command interpreter the user prefers. If there is nothing in the *shell* field, the Bourne shell (/bin/sh) is assumed.

HESIOD SUPPORT

If 'dns' is specified for the 'passwd' database in nsswitch.conf(5), then **passwd** lookups occur from the 'passwd' Hesiod domain.

NIS SUPPORT

If 'nis' is specified for the 'passwd' database in nsswitch.conf(5), then **passwd** lookups occur from the 'passwd.byname', 'passwd.byuid', 'master.passwd.byname', and 'master.passwd.byuid' NIS maps.

COMPAT SUPPORT

If 'compat' is specified for the 'passwd' database, and either 'dns' or 'nis' is specified for the 'passwd_compat' database in nsswitch.conf(5), then the **passwd** file also supports standard '+/-' exclusions and inclusions, based on user names and netgroups.

Lines beginning with a minus sign ("-") are entries marked as being excluded from any following inclusions, which are marked with a plus sign ("+").

If the second character of the line is an at sign ("@"), the operation involves the user fields of all entries in the netgroup specified by the remaining characters of the *name* field. Otherwise, the remainder of the *name* field is assumed to be a specific user name.

The "+" token may also be alone in the *name* field, which causes all users from either the Hesiod domain **passwd** (with 'passwd_compat: dns') or 'passwd.byname' and 'passwd.byuid' NIS maps (with 'passwd_compat: nis') to be included.

If the entry contains non-empty *uid* or *gid* fields, the specified numbers will override the information retrieved from the Hesiod domain or the NIS maps. As well, if the *gecos*, *home_dir* or *shell* entries contain text, it will override the information included via Hesiod or NIS. On some systems, the *passwd* field may also be overridden.

SEE ALSO

chpass(1), login(1), newgrp(1), passwd(1), pwhash(1), getpwent(3), login_getclass(3), login.conf(5), netgroup(5), passwd.conf(5), adduser(8), pwd_mkdb(8), vipw(8), yp(8)

Managing NFS and NIS (O'Reilly & Associates)

COMPATIBILITY

The password file format has changed since 4.3BSD. The following awk script can be used to convert your old-style password file into a new style password file. The additional fields "class", "change" and "expire" are added, but are turned off by default. To set them, use the current day in seconds from the epoch + whatever number of seconds of offset you want.

```
BEGIN { FS = ":"}
{ print $1 ":" $2 ":" $3 ":" $4 "::0:0:" $5 ":" $6 ":" $7 }
```

HISTORY

A **passwd** file format appeared in Version 6 AT&T UNIX.

The NIS **passwd** file format first appeared in SunOS.

The Hesiod support first appeared in NetBSD 1.4.

The `login.conf`(5) capability first appeared in NetBSD 1.5.

BUGS

User information should (and eventually will) be stored elsewhere.

Placing 'compat' exclusions in the file after any inclusions will have unexpected results.

NAME

passwd.conf — password encryption configuration file

SYNOPSIS

passwd.conf

DESCRIPTION

The /etc/passwd.conf file, consisting of "stanzas", describes the configuration of the password cipher used to encrypt local or YP passwords.

There are default, user and group specific stanzas. If no user or group stanza to a specific option is available, the default stanza is used.

To differentiate between user and group stanzas, groups are prefixed with a single colon (':').

Some fields and their possible values that can appear in this file are:

localcipher The cipher to use for local passwords. Possible values are: "old", "newsalt,<rounds>", "md5", "sha1,<rounds>", and "blowfish,<rounds>". For "newsalt" the value of rounds is a 24-bit integer with a minimum of 7250 rounds. For "sha1" the value of rounds is a 32-bit integer, 0 means use the default of 24680. For "blowfish" the value can be between 4 and 31. It specifies the base 2 logarithm of the number of rounds.

ypcipher The cipher to use for YP passwords. The possible values are the same as for localcipher.

To retrieve information from this file use pw_getconf(3).

FILES

/etc/passwd.conf

EXAMPLES

Use MD5 as the local cipher and old-style DES as the YP cipher. Use blowfish with 2^5 rounds for root:

```
default:
     localcipher = md5
     ypcipher = old

root:
     localcipher = blowfish,5
```

SEE ALSO

passwd(1), pwhash(1), pw_getconf(3), passwd(5)

HISTORY

The **passwd.conf** configuration file first appeared in NetBSD 1.6.

NAME

pcre_table − format of Postfix PCRE tables

SYNOPSIS

postmap -q "*string***" pcre:/etc/postfix/***filename*

postmap -q - pcre:/etc/postfix/*filename* <*inputfile*

DESCRIPTION

The Postfix mail system uses optional tables for address rewriting, mail routing, or access control. These tables are usually in **dbm** or **db** format.

Alternatively, lookup tables can be specified in Perl Compatible Regular Expression form. In this case, each input is compared against a list of patterns. When a match is found, the corresponding result is returned and the search is terminated.

To find out what types of lookup tables your Postfix system supports use the "**postconf -m**" command.

To test lookup tables, use the "**postmap -q**" command as described in the SYNOPSIS above.

COMPATIBILITY

With Postfix version 2.2 and earlier specify "**postmap -fq**" to query a table that contains case sensitive patterns. Patterns are case insensitive by default.

TABLE FORMAT

The general form of a PCRE table is:

/*pattern*/*flags result*
: When *pattern* matches the input string, use the corresponding *result* value.

!/*pattern*/*flags result*
: When *pattern* does **not** match the input string, use the corresponding *result* value.

if /*pattern*/*flags*

endif
: Match the input string against the patterns between **if** and **endif**, if and only if that same input string also matches *pattern*. The **if**..**endif** can nest.

  Note: do not prepend whitespace to patterns inside **if**..**endif**.

  This feature is available in Postfix 2.1 and later.

if !/*pattern*/*flags*

endif
: Match the input string against the patterns between **if** and **endif**, if and only if that same input string does **not** match *pattern*. The **if**..**endif** can nest.

  Note: do not prepend whitespace to patterns inside **if**..**endif**.

  This feature is available in Postfix 2.1 and later.

blank lines and comments
: Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

multi-line text
: A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

Each pattern is a perl-like regular expression. The expression delimiter can be any character, except whitespace or characters that have special meaning (traditionally the forward slash is used). The regular expression can contain whitespace.

By default, matching is case-insensitive, and newlines are not treated as special characters. The behavior is controlled by flags, which are toggled by appending one or more of the following characters after the pattern:

i (default: on)
Toggles the case sensitivity flag. By default, matching is case insensitive.

m (default: off)
Toggles the PCRE_MULTILINE flag. When this flag is on, the ^ and **$** metacharacters match immediately after and immediately before a newline character, respectively, in addition to matching at the start and end of the subject string.

s (default: on)
Toggles the PCRE_DOTALL flag. When this flag is on, the **.** metacharacter matches the newline character. With Postfix versions prior to 2.0, The flag is off by default, which is inconvenient for multi-line message header matching.

x (default: off)
Toggles the pcre extended flag. When this flag is on, whitespace in the pattern (other than in a character class) and characters between a **#** outside a character class and the next newline character are ignored. An escaping backslash can be used to include a whitespace or **#** character as part of the pattern.

A (default: off)
Toggles the PCRE_ANCHORED flag. When this flag is on, the pattern is forced to be "anchored", that is, it is constrained to match only at the start of the string which is being searched (the "subject string"). This effect can also be achieved by appropriate constructs in the pattern itself.

E (default: off)
Toggles the PCRE_DOLLAR_ENDONLY flag. When this flag is on, a **$** metacharacter in the pattern matches only at the end of the subject string. Without this flag, a dollar also matches immediately before the final character if it is a newline character (but not before any other newline characters). This flag is ignored if PCRE_MULTILINE flag is set.

U (default: off)
Toggles the ungreedy matching flag. When this flag is on, the pattern matching engine inverts the "greediness" of the quantifiers so that they are not greedy by default, but become greedy if followed by "?". This flag can also set by a (?U) modifier within the pattern.

X (default: off)
Toggles the PCRE_EXTRA flag. When this flag is on, any backslash in a pattern that is followed by a letter that has no special meaning causes an error, thus reserving these combinations for future expansion.

SEARCH ORDER

Patterns are applied in the order as specified in the table, until a pattern is found that matches the input string.

Each pattern is applied to the entire input string. Depending on the application, that string is an entire client hostname, an entire client IP address, or an entire mail address. Thus, no parent domain or parent network search is done, and *user@domain* mail addresses are not broken up into their *user* and *domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

TEXT SUBSTITUTION

Substitution of substrings from the matched expression into the result string is possible using the conventional perl syntax ($1, $2, etc.); specify $$ to produce a $ character as output. The macros in the result string may need to be written as ${n} or $(n) if they aren't followed by whitespace.

Note: since negated patterns (those preceded by **!**) return a result when the expression does not match, substitutions are not available for negated patterns.

EXAMPLE SMTPD ACCESS MAP

```
# Protect your outgoing majordomo exploders
/^(?!owner-)(.*)-outgoing@(.*)/ 550 Use ${1}@${2} instead

# Bounce friend@whatever, except when whatever is our domain (you would
# be better just bouncing all friend@ mail - this is just an example).
/^(friend@(?!my\.domain$).*)$/  550 Stick this in your pipe $1

# A multi-line entry. The text is sent as one line.
#
/^noddy@my\.domain$/
 550 This user is a funny one. You really don't want to send mail to
 them as it only makes their head spin.
```

EXAMPLE HEADER FILTER MAP

```
/^Subject: make money fast/     REJECT
/^To: friend@public\.com/       REJECT
```

EXAMPLE BODY FILTER MAP

```
# First skip over base 64 encoded text to save CPU cycles.
# Requires PCRE version 3.
~^[[:alnum:]+/]{60,}$~          OK

# Put your own body patterns here.
```

SEE ALSO

postmap(1), Postfix lookup table manager
postconf(5), configuration parameters
regexp_table(5), format of POSIX regular expression tables

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview

AUTHOR(S)

The PCRE table lookup code was originally written by:
Andrew McNamara
andrewm@connect.com.au
connect.com.au Pty. Ltd.
Level 3, 213 Miller St
North Sydney, NSW, Australia

Adopted and adapted by:
Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

pf.boot.conf — initial configuration for packet filter

DESCRIPTION

The **pf.boot.conf** file is used as initial configuration for the pf(4) packet filter. This file is loaded before the network is configured by the rc.d(8) script *network*. Its purpose is to protect the machine from possible attacks between the network configuration and the loading of the final ruleset.

The syntax of this file is described in pf.conf(5).

Note that at the stage the configuration is loaded, the network interface(s) do not have an IP address yet, so you *cannot* use rules that derive addresses from an interface (for example: “pass out from any to fxp0”).

FILES

/etc/defaults/pf.boot.conf	Default initial ruleset file.
/etc/pf.boot.conf	Override of the default initial ruleset file.

EXAMPLES

When using NFS (e.g. diskless situations), you’ll also need the following rules in addition to the default rules to unblock NFS:

```
scrub in all no-df
pass in proto udp from any port { 111, 2049 } to any
pass out proto udp from any to any port { 111, 2049 }
```

SEE ALSO

pf(4), pf.conf(5), pfctl(8)

NAME

pf.conf — packet filter configuration file

DESCRIPTION

The pf(4) packet filter modifies, drops or passes packets according to rules or definitions specified in **pf.conf**.

STATEMENT ORDER

There are seven types of statements in **pf.conf**:

Macros
User-defined variables may be defined and used later, simplifying the configuration file. Macros must be defined before they are referenced in **pf.conf**.

Tables
Tables provide a mechanism for increasing the performance and flexibility of rules with large numbers of source or destination addresses.

Options
Options tune the behaviour of the packet filtering engine.

Traffic Normalization (e.g. *scrub*)
Traffic normalization protects internal machines against inconsistencies in Internet protocols and implementations.

Queueing
Queueing provides rule-based bandwidth control.

Translation (Various forms of NAT)
Translation rules specify how addresses are to be mapped or redirected to other addresses.

Packet Filtering
Stateful and stateless packet filtering provides rule-based blocking or passing of packets.

With the exception of **macros** and **tables**, the types of statements should be grouped and appear in **pf.conf** in the order shown above, as this matches the operation of the underlying packet filtering engine. By default pfctl(8) enforces this order (see *set require-order* below).

MACROS

Much like cpp(1) or m4(1), macros can be defined that will later be expanded in context. Macro names must start with a letter, and may contain letters, digits and underscores. Macro names may not be reserved words (for example *pass*, *in*, *out*). Macros are not expanded inside quotes.

For example,

```
ext_if = "kue0"
all_ifs = "{" $ext_if lo0 "}"
pass out on $ext_if from any to any keep state
pass in  on $ext_if proto tcp from any to any port 25 keep state
```

TABLES

Tables are named structures which can hold a collection of addresses and networks. Lookups against tables in pf(4) are relatively fast, making a single rule with tables much more efficient, in terms of processor usage and memory consumption, than a large number of rules which differ only in IP address (either created explicitly or automatically by rule expansion).

Tables can be used as the source or destination of filter rules, *scrub* rules or translation rules such as *nat* or *rdr* (see below for details on the various rule types). Tables can also be used for the redirect address of *nat* and *rdr* rules and in the routing options of filter rules, but only for *round-robin* pools.

Tables can be defined with any of the following pfctl(8) mechanisms. As with macros, reserved words may not be used as table names.

manually Persistent tables can be manually created with the *add* or *replace* option of pfctl(8), before or after the ruleset has been loaded.

pf.conf Table definitions can be placed directly in this file, and loaded at the same time as other rules are loaded, atomically. Table definitions inside **pf.conf** use the *table* statement, and are especially useful to define non-persistent tables. The contents of a pre-existing table defined without a list of addresses to initialize it is not altered when **pf.conf** is loaded. A table initialized with the empty list, { }, will be cleared on load.

Tables may be defined with the following two attributes:

persist The *persist* flag forces the kernel to keep the table even when no rules refer to it. If the flag is not set, the kernel will automatically remove the table when the last rule referring to it is flushed.

const The *const* flag prevents the user from altering the contents of the table once it has been created. Without that flag, pfctl(8) can be used to add or remove addresses from the table at any time, even when running with securelevel = 2.

For example,

```
table <private> const { 10/8, 172.16/12, 192.168/16 }
table <badhosts> persist
block on fxp0 from { <private>, <badhosts> } to any
```

creates a table called private, to hold RFC 1918 private network blocks, and a table called badhosts, which is initially empty. A filter rule is set up to block all traffic coming from addresses listed in either table. The private table cannot have its contents changed and the badhosts table will exist even when no active filter rules reference it. Addresses may later be added to the badhosts table, so that traffic from these hosts can be blocked by using

```
# pfctl -t badhosts -Tadd 204.92.77.111
```

A table can also be initialized with an address list specified in one or more external files, using the following syntax:

```
table <spam> persist file "/etc/spammers" file "/etc/openrelays"
block on fxp0 from <spam> to any
```

The files /etc/spammers and /etc/openrelays list IP addresses, one per line. Any lines beginning with a # are treated as comments and ignored. In addition to being specified by IP address, hosts may also be specified by their hostname. When the resolver is called to add a hostname to a table, *all* resulting IPv4 and IPv6 addresses are placed into the table. IP addresses can also be entered in a table by specifying a valid interface name or the *self* keyword, in which case all addresses assigned to the interface(s) will be added to the table.

OPTIONS

pf(4) may be tuned for various situations using the *set* command.

set timeout

interval Interval between purging expired states and fragments.
frag Seconds before an unassembled fragment is expired.
src.track Length of time to retain a source tracking entry after the last state expires.

When a packet matches a stateful connection, the seconds to live for the connection will be updated to that of the *proto.modifier* which corresponds to the connection state. Each packet which matches this state will reset the TTL. Tuning these values may improve the performance of the firewall at the risk of dropping valid idle connections.

tcp.first
The state after the first packet.
tcp.opening
The state before the destination host ever sends a packet.
tcp.established
The fully established state.
tcp.closing
The state after the first FIN has been sent.
tcp.finwait
The state after both FINs have been exchanged and the connection is closed. Some hosts (notably web servers on Solaris) send TCP packets even after closing the connection. Increasing *tcp.finwait* (and possibly *tcp.closing*) can prevent blocking of such packets.
tcp.closed
The state after one endpoint sends an RST.

ICMP and UDP are handled in a fashion similar to TCP, but with a much more limited set of states:

udp.first
The state after the first packet.
udp.single
The state if the source host sends more than one packet but the destination host has never sent one back.
udp.multiple
The state if both hosts have sent packets.
icmp.first
The state after the first packet.
icmp.error
The state after an ICMP error came back in response to an ICMP packet.

Other protocols are handled similarly to UDP:

other.first
other.single
other.multiple

Timeout values can be reduced adaptively as the number of state table entries grows.

adaptive.start
When the number of state entries exceeds this value, adaptive scaling begins. All timeout values are scaled linearly with factor (adaptive.end - number of states) / (adaptive.end - adaptive.start).
adaptive.end
When reaching this number of state entries, all timeout values become zero, effectively purging all state entries immediately. This value is used to define the scale factor, it should not actually be reached (set a lower state limit, see below).

These values can be defined both globally and for each rule. When used on a per-rule basis, the values relate to the number of states created by the rule, otherwise to the total number of states.

For example:

```
set timeout tcp.first 120
set timeout tcp.established 86400
set timeout { adaptive.start 6000, adaptive.end 12000 }
set limit states 10000
```

With 9000 state table entries, the timeout values are scaled to 50% (tcp.first 60, tcp.established 43200).

set loginterface

Enable collection of packet and byte count statistics for the given interface. These statistics can be viewed using

```
# pfctl -s info
```

In this example pf(4) collects statistics on the interface named dc0:

```
set loginterface dc0
```

One can disable the loginterface using:

```
set loginterface none
```

set limit

Sets hard limits on the memory pools used by the packet filter. See pool(9) for an explanation of memory pools.

For example,

```
set limit states 20000
```

sets the maximum number of entries in the memory pool used by state table entries (generated by *keep state* rules) to 20000. Using

```
set limit frags 20000
```

sets the maximum number of entries in the memory pool used for fragment reassembly (generated by *scrub* rules) to 20000. Finally,

```
set limit src-nodes 2000
```

sets the maximum number of entries in the memory pool used for tracking source IP addresses (generated by the *sticky-address* and *source-track* options) to 2000.

These can be combined:

```
set limit { states 20000, frags 20000, src-nodes 2000 }
```

set optimization

Optimize the engine for one of the following network environments:

normal
: A normal network environment. Suitable for almost all networks.

high-latency
: A high-latency environment (such as a satellite connection).

satellite
: Alias for *high-latency*.

aggressive
Aggressively expire connections. This can greatly reduce the memory usage of the firewall at the cost of dropping idle connections early.

conservative
Extremely conservative settings. Avoid dropping legitimate connections at the expense of greater memory utilization (possibly much greater on a busy network) and slightly increased processor utilization.

For example:

```
set optimization aggressive
```

set block-policy
The *block-policy* option sets the default behaviour for the packet *block* action:

drop Packet is silently dropped.
return A TCP RST is returned for blocked TCP packets, an ICMP UNREACHABLE is returned for blocked UDP packets, and all other packets are silently dropped.

For example:

```
set block-policy return
```

set state-policy
The *state-policy* option sets the default behaviour for states:

if-bound States are bound to interface.
group-bound States are bound to interface group (i.e. ppp)
floating States can match packets on any interfaces (the default).

For example:

```
set state-policy if-bound
```

set require-order
By default `pfctl`(8) enforces an ordering of the statement types in the ruleset to: *options*, *normalization*, *queueing*, *translation*, *filtering*. Setting this option to *no* disables this enforcement. There may be non-trivial and non-obvious implications to an out of order ruleset. Consider carefully before disabling the order enforcement.

set fingerprints
Load fingerprints of known operating systems from the given filename. By default fingerprints of known operating systems are automatically loaded from `pf.os`(5) in `/etc` but can be overridden via this option. Setting this option may leave a small period of time where the fingerprints referenced by the currently active ruleset are inconsistent until the new ruleset finishes loading.

For example:

```
set fingerprints "/etc/pf.os.devel"
```

set skip on <ifspec>
List interfaces for which packets should not be filtered. Packets passing in or out on such interfaces are passed as if pf was disabled, i.e. pf does not process them in any way. This can be useful on loopback and other virtual interfaces, when packet filtering is not desired and can have unexpected effects. For example:

```
set skip on lo0
```

set debug

Set the debug *level* to one of the following:

none	Don't generate debug messages.
urgent	Generate debug messages only for serious errors.
misc	Generate debug messages for various errors.
loud	Generate debug messages for common conditions.

TRAFFIC NORMALIZATION

Traffic normalization is used to sanitize packet content in such a way that there are no ambiguities in packet interpretation on the receiving side. The normalizer does IP fragment reassembly to prevent attacks that confuse intrusion detection systems by sending overlapping IP fragments. Packet normalization is invoked with the *scrub* directive.

scrub has the following options:

no-df

Clears the *dont-fragment* bit from a matching IP packet. Some operating systems are known to generate fragmented packets with the *dont-fragment* bit set. This is particularly true with NFS. *Scrub* will drop such fragmented *dont-fragment* packets unless *no-df* is specified.

Unfortunately some operating systems also generate their *dont-fragment* packets with a zero IP identification field. Clearing the *dont-fragment* bit on packets with a zero IP ID may cause deleterious results if an upstream router later fragments the packet. Using the *random-id* modifier (see below) is recommended in combination with the *no-df* modifier to ensure unique IP identifiers.

min-ttl <number>

Enforces a minimum TTL for matching IP packets.

max-mss <number>

Enforces a maximum MSS for matching TCP packets.

random-id

Replaces the IP identification field with random values to compensate for predictable values generated by many hosts. This option only applies to packets that are not fragmented after the optional fragment reassembly.

fragment reassemble

Using *scrub* rules, fragments can be reassembled by normalization. In this case, fragments are buffered until they form a complete packet, and only the completed packet is passed on to the filter. The advantage is that filter rules have to deal only with complete packets, and can ignore fragments. The drawback of caching fragments is the additional memory cost. But the full reassembly method is the only method that currently works with NAT. This is the default behavior of a *scrub* rule if no fragmentation modifier is supplied.

fragment crop

The default fragment reassembly method is expensive, hence the option to crop is provided. In this case, pf(4) will track the fragments and cache a small range descriptor. Duplicate fragments are dropped and overlaps are cropped. Thus data will only occur once on the wire with ambiguities resolving to the first occurrence. Unlike the *fragment reassemble* modifier, fragments are not buffered, they are passed as soon as they are received. The *fragment crop* reassembly mechanism does not yet work with NAT.

fragment drop-ovl

This option is similar to the *fragment crop* modifier except that all overlapping or duplicate fragments will be dropped, and all further corresponding fragments will be dropped as well.

reassemble tcp

Statefully normalizes TCP connections. *scrub reassemble tcp* rules may not have the direction (in/out) specified. *reassemble tcp* performs the following normalizations:

ttl
: Neither side of the connection is allowed to reduce their IP TTL. An attacker may send a packet such that it reaches the firewall, affects the firewall state, and expires before reaching the destination host. *reassemble tcp* will raise the TTL of all packets back up to the highest value seen on the connection.

timestamp modulation
: Modern TCP stacks will send a timestamp on every TCP packet and echo the other endpoint's timestamp back to them. Many operating systems will merely start the timestamp at zero when first booted, and increment it several times a second. The uptime of the host can be deduced by reading the timestamp and multiplying by a constant. Also observing several different timestamps can be used to count hosts behind a NAT device. And spoofing TCP packets into a connection requires knowing or guessing valid timestamps. Timestamps merely need to be monotonically increasing and not derived off a guessable base time. *reassemble tcp* will cause *scrub* to modulate the TCP timestamps with a random number.

extended PAWS checks
: There is a problem with TCP on long fat pipes, in that a packet might get delayed for longer than it takes the connection to wrap its 32-bit sequence space. In such an occurrence, the old packet would be indistinguishable from a new packet and would be accepted as such. The solution to this is called PAWS: Protection Against Wrapped Sequence numbers. It protects against it by making sure the timestamp on each packet does not go backwards. *reassemble tcp* also makes sure the timestamp on the packet does not go forward more than the RFC allows. By doing this, pf(4) artificially extends the security of TCP sequence numbers by 10 to 18 bits when the host uses appropriately randomized timestamps, since a blind attacker would have to guess the timestamp as well.

For example,

```
scrub in on $ext_if all fragment reassemble
```

The *no* option prefixed to a scrub rule causes matching packets to remain unscrubbed, much in the same way as *drop quick* works in the packet filter (see below). This mechanism should be used when it is necessary to exclude specific packets from broader scrub rules.

QUEUEING

Packets can be assigned to queues for the purpose of bandwidth control. At least two declarations are required to configure queues, and later any packet filtering rule can reference the defined queues by name. During the filtering component of **pf.conf**, the last referenced *queue* name is where any packets from *pass* rules will be queued, while for *block* rules it specifies where any resulting ICMP or TCP RST packets should be queued. The *scheduler* defines the algorithm used to decide which packets get delayed, dropped, or sent out immediately. There are three *schedulers* currently supported.

cbq Class Based Queueing. *Queues* attached to an interface build a tree, thus each *queue* can have further child *queues*. Each queue can have a *priority* and a *bandwidth* assigned. *Priority* mainly controls the time packets take to get sent out, while *bandwidth* has primarily effects on throughput. *cbq* achieves both partitioning and sharing of link bandwidth by hierarchically structured classes. Each class has its own *queue* and is assigned its share of *bandwidth*. A child class can borrow bandwidth from its parent class as long as excess bandwidth is available (see the option *borrow*, below).

priq Priority Queueing. *Queues* are flat attached to the interface, thus, *queues* cannot have further child *queues*. Each *queue* has a unique *priority* assigned, ranging from 0 to 15. Packets in the *queue* with the highest *priority* are processed first.

hfsc Hierarchical Fair Service Curve. *Queues* attached to an interface build a tree, thus each *queue* can have further child *queues*. Each queue can have a *priority* and a *bandwidth* assigned. *Priority* mainly controls the time packets take to get sent out, while *bandwidth* has primarily effects on throughput. *hfsc* supports both link-sharing and guaranteed real-time services. It employs a service curve based QoS model, and its unique feature is an ability to decouple *delay* and *bandwidth* allocation.

The interfaces on which queueing should be activated are declared using the *altq on* declaration. *altq on* has the following keywords:

<interface>
Queueing is enabled on the named interface.

<scheduler>
Specifies which queueing scheduler to use. Currently supported values are *cbq* for Class Based Queueing, *priq* for Priority Queueing and *hfsc* for the Hierarchical Fair Service Curve scheduler.

bandwidth <bw>
The maximum bitrate for all queues on an interface may be specified using the *bandwidth* keyword. The value can be specified as an absolute value or as a percentage of the interface bandwidth. When using an absolute value, the suffixes *b*, *Kb*, *Mb*, and *Gb* are used to represent bits, kilobits, megabits, and gigabits per second, respectively. The value must not exceed the interface bandwidth. If *bandwidth* is not specified, the interface bandwidth is used.

qlimit <limit>
The maximum number of packets held in the queue. The default is 50.

tbrsize <size>
Adjusts the size, in bytes, of the token bucket regulator. If not specified, heuristics based on the interface bandwidth are used to determine the size.

queue <list>
Defines a list of subqueues to create on an interface.

In the following example, the interface dc0 should queue up to 5 Mbit/s in four second-level queues using Class Based Queueing. Those four queues will be shown in a later example.

```
altq on dc0 cbq bandwidth 5Mb queue { std, http, mail, ssh }
```

Once interfaces are activated for queueing using the *altq* directive, a sequence of *queue* directives may be defined. The name associated with a *queue* must match a queue defined in the *altq* directive (e.g. mail), or, except for the *priq scheduler*, in a parent *queue* declaration. The following keywords can be used:

on <interface>
Specifies the interface the queue operates on. If not given, it operates on all matching interfaces.

bandwidth <bw>
Specifies the maximum bitrate to be processed by the queue. This value must not exceed the value of the parent *queue* and can be specified as an absolute value or a percentage of the parent queue's bandwidth. If not specified, defaults to 100% of the parent queue's bandwidth. The *priq* scheduler does not support bandwidth specification.

priority <level>
Between queues a priority level can be set. For *cbq* and *hfsc*, the range is 0 to 7 and for *priq*, the range is 0 to 15. The default for all is 1. *Priq* queues with a higher priority are always served first.

Cbq and *Hfsc* queues with a higher priority are preferred in the case of overload.

qlimit <limit>
The maximum number of packets held in the queue. The default is 50.

The *scheduler* can get additional parameters with *<scheduler>(<parameters>*). Parameters are as follows:

default Packets not matched by another queue are assigned to this one. Exactly one default queue is required.

red Enable RED (Random Early Detection) on this queue. RED drops packets with a probability proportional to the average queue length.

rio Enables RIO on this queue. RIO is RED with IN/OUT, thus running RED two times more than RIO would achieve the same effect. RIO is currently not supported in the GENERIC kernel.

ecn Enables ECN (Explicit Congestion Notification) on this queue. ECN implies RED.

The *cbq scheduler* supports an additional option:

borrow The queue can borrow bandwidth from the parent.

The *hfsc scheduler* supports some additional options:

realtime <sc>
The minimum required bandwidth for the queue.

upperlimit <sc>
The maximum allowed bandwidth for the queue.

linkshare <sc>
The bandwidth share of a backlogged queue.

<sc> is an acronym for *service curve*.

The format for service curve specifications is (*m1*, *d*, *m2*). *m2* controls the bandwidth assigned to the queue. *m1* and *d* are optional and can be used to control the initial bandwidth assignment. For the first *d* milliseconds the queue gets the bandwidth given as *m1*, afterwards the value given in *m2*.

Furthermore, with *cbq* and *hfsc*, child queues can be specified as in an *altq* declaration, thus building a tree of queues using a part of their parent's bandwidth.

Packets can be assigned to queues based on filter rules by using the *queue* keyword. Normally only one *queue* is specified; when a second one is specified it will instead be used for packets which have a *TOS* of *lowdelay* and for TCP ACKs with no data payload.

To continue the previous example, the examples below would specify the four referenced queues, plus a few child queues. Interactive ssh(1) sessions get priority over bulk transfers like scp(1) and sftp(1). The queues may then be referenced by filtering rules (see **PACKET FILTERING** below).

```
queue std bandwidth 10% cbq(default)
queue http bandwidth 60% priority 2 cbq(borrow red) \
      { employees, developers }
queue  developers bandwidth 75% cbq(borrow)
queue  employees bandwidth 15%
queue mail bandwidth 10% priority 0 cbq(borrow ecn)
queue ssh bandwidth 20% cbq(borrow) { ssh_interactive, ssh_bulk }
queue  ssh_interactive bandwidth 50% priority 7 cbq(borrow)
queue  ssh_bulk bandwidth 50% priority 0 cbq(borrow)
```

```
block return out on dc0 inet all queue std
pass out on dc0 inet proto tcp from $developerhosts to any port 80 \
      keep state queue developers
pass out on dc0 inet proto tcp from $employeehosts to any port 80 \
      keep state queue employees
pass out on dc0 inet proto tcp from any to any port 22 \
      keep state queue(ssh_bulk, ssh_interactive)
pass out on dc0 inet proto tcp from any to any port 25 \
      keep state queue mail
```

TRANSLATION

Translation rules modify either the source or destination address of the packets associated with a stateful connection. A stateful connection is automatically created to track packets matching such a rule as long as they are not blocked by the filtering section of **pf.conf**. The translation engine modifies the specified address and/or port in the packet, recalculates IP, TCP and UDP checksums as necessary, and passes it to the packet filter for evaluation.

Since translation occurs before filtering the filter engine will see packets as they look after any addresses and ports have been translated. Filter rules will therefore have to filter based on the translated address and port number. Packets that match a translation rule are only automatically passed if the *pass* modifier is given, otherwise they are still subject to *block* and *pass* rules.

The state entry created permits pf(4) to keep track of the original address for traffic associated with that state and correctly direct return traffic for that connection.

Various types of translation are possible with pf:

binat

A *binat* rule specifies a bidirectional mapping between an external IP netblock and an internal IP netblock.

nat A *nat* rule specifies that IP addresses are to be changed as the packet traverses the given interface. This technique allows one or more IP addresses on the translating host to support network traffic for a larger range of machines on an "inside" network. Although in theory any IP address can be used on the inside, it is strongly recommended that one of the address ranges defined by RFC 1918 be used. These netblocks are:

```
10.0.0.0 - 10.255.255.255 (all of net 10, i.e., 10/8)
172.16.0.0 - 172.31.255.255 (i.e., 172.16/12)
192.168.0.0 - 192.168.255.255 (i.e., 192.168/16)
```

rdr The packet is redirected to another destination and possibly a different port. *rdr* rules can optionally specify port ranges instead of single ports. rdr ... port 2000:2999 -> ... port 4000 redirects ports 2000 to 2999 (inclusive) to port 4000. rdr ... port 2000:2999 -> ... port 4000:* redirects port 2000 to 4000, 2001 to 4001, ..., 2999 to 4999.

In addition to modifying the address, some translation rules may modify source or destination ports for tcp(4) or udp(4) connections; implicitly in the case of *nat* rules and explicitly in the case of *rdr* rules. Port numbers are never translated with a *binat* rule.

For each packet processed by the translator, the translation rules are evaluated in sequential order, from first to last. The first matching rule decides what action is taken.

The *no* option prefixed to a translation rule causes packets to remain untranslated, much in the same way as *drop quick* works in the packet filter (see below). If no rule matches the packet it is passed to the filter engine unmodified.

Translation rules apply only to packets that pass through the specified interface, and if no interface is specified, translation is applied to packets on all interfaces. For instance, redirecting port 80 on an external interface to an internal web server will only work for connections originating from the outside. Connections to the address of the external interface from local hosts will not be redirected, since such packets do not actually pass through the external interface. Redirections cannot reflect packets back through the interface they arrive on, they can only be redirected to hosts connected to different interfaces or to the firewall itself.

Note that redirecting external incoming connections to the loopback address, as in

```
rdr on ne3 inet proto tcp to port 8025 -> 127.0.0.1 port 25
```

will effectively allow an external host to connect to daemons bound solely to the loopback address, circumventing the traditional blocking of such connections on a real interface. Unless this effect is desired, any of the local non-loopback addresses should be used as redirection target instead, which allows external connections only to daemons bound to this address or not bound to any address.

See **TRANSLATION EXAMPLES** below.

PACKET FILTERING

pf(4) has the ability to *block* and *pass* packets based on attributes of their layer 3 (see ip(4) and ip6(4)) and layer 4 (see icmp(4), icmp6(4), tcp(4), udp(4)) headers. In addition, packets may also be assigned to queues for the purpose of bandwidth control.

For each packet processed by the packet filter, the filter rules are evaluated in sequential order, from first to last. The last matching rule decides what action is taken.

The following actions can be used in the filter:

block
: The packet is blocked. There are a number of ways in which a *block* rule can behave when blocking a packet. The default behaviour is to *drop* packets silently, however this can be overridden or made explicit either globally, by setting the *block-policy* option, or on a per-rule basis with one of the following options:

  drop The packet is silently dropped.

  return-rst
  : This applies only to tcp(4) packets, and issues a TCP RST which closes the connection.

  return-icmp

  return-icmp6
  : This causes ICMP messages to be returned for packets which match the rule. By default this is an ICMP UNREACHABLE message, however this can be overridden by specifying a message as a code or number.

  return
  : This causes a TCP RST to be returned for tcp(4) packets and an ICMP UNREACHABLE for UDP and other packets.

  Options returning ICMP packets currently have no effect if pf(4) operates on a bridge(4), as the code to support this feature has not yet been implemented.

pass The packet is passed.

If no rule matches the packet, the default action is *pass*.

To block everything by default and only pass packets that match explicit rules, one uses

```
block all
```

as the first filter rule.

See **FILTER EXAMPLES** below.

PARAMETERS

The rule parameters specify the packets to which a rule applies. A packet always comes in on, or goes out through, one interface. Most parameters are optional. If a parameter is specified, the rule only applies to packets with matching attributes. Certain parameters can be expressed as lists, in which case `pfctl`(8) generates all needed rule combinations.

in or *out*
: This rule applies to incoming or outgoing packets. If neither *in* nor *out* are specified, the rule will match packets in both directions.

log
: In addition to the action specified, a log message is generated. All packets for that connection are logged, unless the *keep state*, *modulate state* or *synproxy state* options are specified, in which case only the packet that establishes the state is logged. (See *keep state*, *modulate state* and *synproxy state* below). The logged packets are sent to the `pflog`(4) interface. This interface is monitored by the `pflogd`(8) logging daemon, which dumps the logged packets to the file `/var/log/pflog` in `pcap`(3) binary format.

log-all
: Used with *keep state*, *modulate state* or *synproxy state* rules to force logging of all packets for a connection. As with *log*, packets are logged to `pflog`(4).

quick
: If a packet matches a rule which has the *quick* option set, this rule is considered the last matching rule, and evaluation of subsequent rules is skipped.

on <interface>
: This rule applies only to packets coming in on, or going out through, this particular interface. It is also possible to simply give the interface driver name, like ppp or fxp, to make the rule match packets flowing through a group of interfaces.

<af> This rule applies only to packets of this address family. Supported values are *inet* and *inet6*.

proto <protocol>
: This rule applies only to packets of this protocol. Common protocols are `icmp`(4), `icmp6`(4), `tcp`(4), and `udp`(4). For a list of all the protocol name to number mappings used by `pfctl`(8), see the file */etc/protocols*.

from <source> port <source> os <source> to <dest> port <dest>
: This rule applies only to packets with the specified source and destination addresses and ports.

  Addresses can be specified in CIDR notation (matching netblocks), as symbolic host names or interface names, or as any of the following keywords:

any	Any address.
route <label>	Any address whose associated route has label *<label>*. See `route`(4) and `route`(8).
no-route	Any address which is not currently routable.
<table>	Any address that matches the given table.

  Interface names can have modifiers appended:

:network	Translates to the network(s) attached to the interface.

:broadcast	Translates to the interface's broadcast address(es).
:peer	Translates to the point to point interface's peer address(es).
:0	Do not include interface aliases.

Host names may also have the *:0* option appended to restrict the name resolution to the first of each v4 and v6 address found.

Host name resolution and interface to address translation are done at ruleset load-time. When the address of an interface (or host name) changes (under DHCP or PPP, for instance), the ruleset must be reloaded for the change to be reflected in the kernel. Surrounding the interface name (and optional modifiers) in parentheses changes this behaviour. When the interface name is surrounded by parentheses, the rule is automatically updated whenever the interface changes its address. The ruleset does not need to be reloaded. This is especially useful with *nat*.

Ports can be specified either by number or by name. For example, port 80 can be specified as *www*. For a list of all port name to number mappings used by pfctl(8), see the file /etc/services.

Ports and ranges of ports are specified by using these operators:

```
=       (equal)
!=      (unequal)
<       (less than)
<=      (less than or equal)
>       (greater than)
>=      (greater than or equal)
:       (range including boundaries)
><      (range excluding boundaries)
<>      (except range)
```

><, <> and : are binary operators (they take two arguments). For instance:

port 2000:2004
means 'all ports >= 2000 and <= 2004', hence ports 2000, 2001, 2002, 2003 and 2004.

port 2000 >< 2004
means 'all ports > 2000 and < 2004', hence ports 2001, 2002 and 2003.

port 2000 <> 2004
means 'all ports < 2000 or > 2004', hence ports 1-1999 and 2005-65535.

The operating system of the source host can be specified in the case of TCP rules with the *OS* modifier. See the **OPERATING SYSTEM FINGERPRINTING** section for more information.

The host, port and OS specifications are optional, as in the following examples:

```
pass in all
pass in from any to any
pass in proto tcp from any port <= 1024 to any
pass in proto tcp from any to any port 25
pass in proto tcp from 10.0.0.0/8 port > 1024 \
      to ! 10.1.2.3 port != ssh
pass in proto tcp from any os "OpenBSD" flags S/SA
pass in proto tcp from route "DTAG"
```

all This is equivalent to "from any to any".

group <group>
This functionality is not supported in this version of NetBSD.

user <user>

This rule only applies to packets of sockets owned by the specified user. For outgoing connections initiated from the firewall, this is the user that opened the connection. For incoming connections to the firewall itself, this is the user that listens on the destination port. For forwarded connections, where the firewall is not a connection endpoint, the user and group are *unknown*.

All packets, both outgoing and incoming, of one connection are associated with the same user and group. Only TCP and UDP packets can be associated with users; for other protocols these parameters are ignored.

User and group refer to the effective (as opposed to the real) IDs, in case the socket is created by a setuid/setgid process. User and group IDs are stored when a socket is created; when a process creates a listening socket as root (for instance, by binding to a privileged port) and subsequently changes to another user ID (to drop privileges), the credentials will remain root.

User and group IDs can be specified as either numbers or names. The syntax is similar to the one for ports. The value *unknown* matches packets of forwarded connections. *unknown* can only be used with the operators **=** and **!=**. Other constructs like **user >= unknown** are invalid. Forwarded packets with unknown user and group ID match only rules that explicitly compare against *unknown* with the operators **=** or **!=**. For instance **user >= 0** does not match forwarded packets. The following example allows only selected users to open outgoing connections:

```
block out proto { tcp, udp } all
pass  out proto { tcp, udp } all \
      user { < 1000, dhartmei } keep state
```

flags <a>/<b> | */<b>*

This rule only applies to TCP packets that have the flags *<a>* set out of set *<b>*. Flags not specified in *<b>* are ignored. The flags are: (F)IN, (S)YN, (R)ST, (P)USH, (A)CK, (U)RG, (E)CE, and C(W)R.

flags S/S

Flag SYN is set. The other flags are ignored.

flags S/SA

Out of SYN and ACK, exactly SYN may be set. SYN, SYN+PSH and SYN+RST match, but SYN+ACK, ACK and ACK+RST do not. This is more restrictive than the previous example.

flags /SFRA

If the first set is not specified, it defaults to none. All of SYN, FIN, RST and ACK must be unset.

icmp-type <type> code <code>

icmp6-type <type> code <code>

This rule only applies to ICMP or ICMPv6 packets with the specified type and code. Text names for ICMP types and codes are listed in icmp(4) and icmp6(4). This parameter is only valid for rules that cover protocols ICMP or ICMP6. The protocol and the ICMP type indicator (*icmp-type* or *icmp6-type*) must match.

tos ⟨*string*⟩ | ⟨*number*⟩

This rule applies to packets with the specified *TOS* bits set. *TOS* may be given as one of *lowdelay*, *throughput*, *reliability*, or as either hex or decimal.

For example, the following rules are identical:

```
pass all tos lowdelay
pass all tos 0x10
pass all tos 16
```

allow-opts
By default, packets which contain IP options are blocked. When *allow-opts* is specified for a *pass* rule, packets that pass the filter based on that rule (last matching) do so even if they contain IP options. For packets that match state, the rule that initially created the state is used. The implicit *pass* rule that is used when a packet does not match any rules does not allow IP options.

label <string>
Adds a label (name) to the rule, which can be used to identify the rule. For instance, pfctl -s labels shows per-rule statistics for rules that have labels.

The following macros can be used in labels:

$if	The interface.
$srcaddr	The source IP address.
$dstaddr	The destination IP address.
$srcport	The source port specification.
$dstport	The destination port specification.
$proto	The protocol name.
$nr	The rule number.

For example:

```
ips = "{ 1.2.3.4, 1.2.3.5 }"
pass in proto tcp from any to $ips \
      port > 1023 label "$dstaddr:$dstport"
```

expands to

```
pass in inet proto tcp from any to 1.2.3.4 \
      port > 1023 label "1.2.3.4:>1023"
pass in inet proto tcp from any to 1.2.3.5 \
      port > 1023 label "1.2.3.5:>1023"
```

The macro expansion for the *label* directive occurs only at configuration file parse time, not during runtime.

queue <queue> | (*<queue>*, *<queue>*)
Packets matching this rule will be assigned to the specified queue. If two queues are given, packets which have a *TOS* of *lowdelay* and TCP ACKs with no data payload will be assigned to the second one. See **QUEUEING** for setup details.

For example:

```
pass in proto tcp to port 25 queue mail
pass in proto tcp to port 22 queue(ssh_bulk, ssh_prio)
```

tag <string>
Packets matching this rule will be tagged with the specified string. The tag acts as an internal marker that can be used to identify these packets later on. This can be used, for example, to provide trust between interfaces and to determine if packets have been processed by translation rules. Tags are "sticky", meaning that the packet will be tagged even if the rule is not the last matching rule. Further matching rules can replace the tag with a new one but will not remove a previously applied tag. A packet is only ever assigned one tag at a time. *pass* rules that use the *tag* keyword must also use *keep state*, *modulate state* or *synproxy state*. Packet tagging can be done during

nat, *rdr*, or *binat* rules in addition to filter rules. Tags take the same macros as labels (see above).

tagged <string>
Used with filter or translation rules to specify that packets must already be tagged with the given tag in order to match the rule. Inverse tag matching can also be done by specifying the **!** operator before the *tagged* keyword.

probability <number>
A probability attribute can be attached to a rule, with a value set between 0 and 1, bounds not included. In that case, the rule will be honoured using the given probability value only. For example, the following rule will drop 20% of incoming ICMP packets:

```
block in proto icmp probability 20%
```

ROUTING

If a packet matches a rule with a route option set, the packet filter will route the packet according to the type of route option. When such a rule creates state, the route option is also applied to all packets matching the same connection.

fastroute
The *fastroute* option does a normal route lookup to find the next hop for the packet.

route-to
The *route-to* option routes the packet to the specified interface with an optional address for the next hop. When a *route-to* rule creates state, only packets that pass in the same direction as the filter rule specifies will be routed in this way. Packets passing in the opposite direction (replies) are not affected and are routed normally.

reply-to
The *reply-to* option is similar to *route-to*, but routes packets that pass in the opposite direction (replies) to the specified interface. Opposite direction is only defined in the context of a state entry, and *reply-to* is useful only in rules that create state. It can be used on systems with multiple external connections to route all outgoing packets of a connection through the interface the incoming connection arrived through (symmetric routing enforcement).

dup-to
The *dup-to* option creates a duplicate of the packet and routes it like *route-to*. The original packet gets routed as it normally would.

POOL OPTIONS

For *nat* and *rdr* rules, (as well as for the *route-to*, *reply-to* and *dup-to* rule options) for which there is a single redirection address which has a subnet mask smaller than 32 for IPv4 or 128 for IPv6 (more than one IP address), a variety of different methods for assigning this address can be used:

bitmask
The *bitmask* option applies the network portion of the redirection address to the address to be modified (source with *nat*, destination with *rdr*).

random
The *random* option selects an address at random within the defined block of addresses.

source-hash
The *source-hash* option uses a hash of the source address to determine the redirection address, ensuring that the redirection address is always the same for a given source. An optional key can be specified after this keyword either in hex or as a string; by default `pfctl`(8) randomly generates a key for source-hash every time the ruleset is reloaded.

round-robin
The *round-robin* option loops through the redirection address(es).

When more than one redirection address is specified, *round-robin* is the only permitted pool type.

static-port
With *nat* rules, the *static-port* option prevents pf(4) from modifying the source port on TCP and UDP packets.

Additionally, the *sticky-address* option can be specified to help ensure that multiple connections from the same source are mapped to the same redirection address. This option can be used with the *random* and *round-robin* pool options. Note that by default these associations are destroyed as soon as there are no longer states which refer to them; in order to make the mappings last beyond the lifetime of the states, increase the global options with *set timeout source-track* See **STATEFUL TRACKING OPTIONS** for more ways to control the source tracking.

STATEFUL INSPECTION

pf(4) is a stateful packet filter, which means it can track the state of a connection. Instead of passing all traffic to port 25, for instance, it is possible to pass only the initial packet, and then begin to keep state. Subsequent traffic will flow because the filter is aware of the connection.

If a packet matches a *pass ... keep state* rule, the filter creates a state for this connection and automatically lets pass all subsequent packets of that connection.

Before any rules are evaluated, the filter checks whether the packet matches any state. If it does, the packet is passed without evaluation of any rules.

States are removed after the connection is closed or has timed out.

This has several advantages. Comparing a packet to a state involves checking its sequence numbers. If the sequence numbers are outside the narrow windows of expected values, the packet is dropped. This prevents spoofing attacks, such as when an attacker sends packets with a fake source address/port but does not know the connection's sequence numbers.

Also, looking up states is usually faster than evaluating rules. If there are 50 rules, all of them are evaluated sequentially in O(n). Even with 50000 states, only 16 comparisons are needed to match a state, since states are stored in a binary search tree that allows searches in O(log2 n).

For instance:

```
block all
pass out proto tcp from any to any flags S/SA keep state
pass in  proto tcp from any to any port 25 flags S/SA keep state
```

This ruleset blocks everything by default. Only outgoing connections and incoming connections to port 25 are allowed. The initial packet of each connection has the SYN flag set, will be passed and creates state. All further packets of these connections are passed if they match a state.

By default, packets coming in and out of any interface can match a state, but it is also possible to change that behaviour by assigning states to a single interface or a group of interfaces.

The default policy is specified by the *state-policy* global option, but this can be adjusted on individual filter rules by adding one of the *if-bound*, *group-bound*, or *floating* keywords to the *keep state* option. For example, if a rule is defined as:

```
pass out on ppp from any to 10.12/16 keep state (group-bound)
```

A state created on ppp0 would match packets an all PPP interfaces, but not packets flowing through fxp0 or any other interface.

You can adjust the state policy on individual *nat* and *rdr* translation rules by adding a keyword *if-bound*, *group-bound* or *floating* at the end of the rule. For example, a rule such as this,

```
nat on sip0 from 10/8 to ! 10/8 -> 192.168.1.4/32 if-bound
```

will create states that only match packets on sip0.

Keeping rules *floating* is the more flexible option when the firewall is in a dynamic routing environment. However, this has some security implications since a state created by one trusted network could allow potentially hostile packets coming in from other interfaces.

Specifying *flags S/SA* restricts state creation to the initial SYN packet of the TCP handshake. One can also be less restrictive, and allow state creation from intermediate (non-SYN) packets. This will cause pf(4) to synchronize to existing connections, for instance if one flushes the state table.

For UDP, which is stateless by nature, *keep state* will create state as well. UDP packets are matched to states using only host addresses and ports.

ICMP messages fall into two categories: ICMP error messages, which always refer to a TCP or UDP packet, are matched against the referred to connection. If one keeps state on a TCP connection, and an ICMP source quench message referring to this TCP connection arrives, it will be matched to the right state and get passed.

For ICMP queries, *keep state* creates an ICMP state, and pf(4) knows how to match ICMP replies to states. For example,

```
pass out inet proto icmp all icmp-type echoreq keep state
```

allows echo requests (such as those created by ping(8)) out, creates state, and matches incoming echo replies correctly to states.

Note: *nat*, *binat* and *rdr* rules implicitly create state for connections.

STATE MODULATION

Much of the security derived from TCP is attributable to how well the initial sequence numbers (ISNs) are chosen. Some popular stack implementations choose *very* poor ISNs and thus are normally susceptible to ISN prediction exploits. By applying a *modulate state* rule to a TCP connection, pf(4) will create a high quality random sequence number for each connection endpoint.

The *modulate state* directive implicitly keeps state on the rule and is only applicable to TCP connections.

For instance:

```
block all
pass out proto tcp from any to any modulate state
pass in  proto tcp from any to any port 25 flags S/SA modulate state
```

There are two caveats associated with state modulation: A *modulate state* rule can not be applied to a pre-existing but unmodulated connection. Such an application would desynchronize TCP's strict sequencing between the two endpoints. Instead, pf(4) will treat the *modulate state* modifier as a *keep state* modifier and the pre-existing connection will be inferred without the protection conferred by modulation.

The other caveat affects currently modulated states when the state table is lost (firewall reboot, flushing the state table, etc...). pf(4) will not be able to infer a connection again after the state table flushes the connection's modulator. When the state is lost, the connection may be left dangling until the respective endpoints time out the connection. It is possible on a fast local network for the endpoints to start an ACK storm while trying to resynchronize after the loss of the modulator. Using a *flags S/SA* modifier on *modulate state* rules between fast networks is suggested to prevent ACK storms.

SYN PROXY

By default, pf(4) passes packets that are part of a tcp(4) handshake between the endpoints. The *synproxy state* option can be used to cause pf(4) itself to complete the handshake with the active endpoint, perform a handshake with the passive endpoint, and then forward packets between the endpoints.

No packets are sent to the passive endpoint before the active endpoint has completed the handshake, hence so-called SYN floods with spoofed source addresses will not reach the passive endpoint, as the sender can't complete the handshake.

The proxy is transparent to both endpoints, they each see a single connection from/to the other endpoint. pf(4) chooses random initial sequence numbers for both handshakes. Once the handshakes are completed, the sequence number modulators (see previous section) are used to translate further packets of the connection. Hence, *synproxy state* includes *modulate state* and *keep state*.

Rules with *synproxy* will not work if pf(4) operates on a bridge(4).

Example:

```
pass in proto tcp from any to any port www flags S/SA synproxy state
```

STATEFUL TRACKING OPTIONS

All three of *keep state*, *modulate state* and *synproxy state* support the following options:

max <number>
: Limits the number of concurrent states the rule may create. When this limit is reached, further packets matching the rule that would create state are dropped, until existing states time out.

<timeout> <seconds>
: Changes the timeout values used for states created by this rule. For a list of all valid timeout names, see **OPTIONS** above.

Multiple options can be specified, separated by commas:

```
pass in proto tcp from any to any \
      port www flags S/SA keep state \
      (max 100, source-track rule, max-src-nodes 75, \
      max-src-states 3, tcp.established 60, tcp.closing 5)
```

When the *source-track* keyword is specified, the number of states per source IP is tracked.

source-track rule
: The maximum number of states created by this rule is limited by the rule's *max-src-nodes* and *max-src-state* options. Only state entries created by this particular rule count toward the rule's limits.

source-track global
: The number of states created by all rules that use this option is limited. Each rule can specify different *max-src-nodes* and *max-src-states* options, however state entries created by any participating rule count towards each individual rule's limits.

The following limits can be set:

max-src-nodes <number>
: Limits the maximum number of source addresses which can simultaneously have state table entries.

max-src-states <number>
: Limits the maximum number of simultaneous state entries that a single source address can create with this rule.

For stateful TCP connections, limits on established connections (connections which have completed the TCP 3-way handshake) can also be enforced per source IP.

max-src-conn <number>
Limits the maximum number of simultaneous TCP connections which have completed the 3-way handshake that a single host can make.

max-src-conn-rate <number> / <seconds>
Limit the rate of new connections over a time interval. The connection rate is an approximation calculated as a moving average.

Because the 3-way handshake ensures that the source address is not being spoofed, more aggressive action can be taken based on these limits. With the *overload <table>* state option, source IP addresses which hit either of the limits on established connections will be added to the named table. This table can be used in the ruleset to block further activity from the offending host, redirect it to a tarpit process, or restrict its bandwidth.

The optional *flush* keyword kills all states created by the matching rule which originate from the host which exceeds these limits. The *global* modifier to the flush command kills all states originating from the offending host, regardless of which rule created the state.

For example, the following rules will protect the webserver against hosts making more than 100 connections in 10 seconds. Any host which connects faster than this rate will have its address added to the <bad_hosts> table and have all states originating from it flushed. Any new packets arriving from this host will be dropped unconditionally by the block rule.

```
block quick from <bad_hosts>
pass in on $ext_if proto tcp to $webserver port www flags S/SA keep state \
       (max-src-conn-rate 100/10, overload <bad_hosts> flush global)
```

OPERATING SYSTEM FINGERPRINTING

Passive OS Fingerprinting is a mechanism to inspect nuances of a TCP connection's initial SYN packet and guess at the host's operating system. Unfortunately these nuances are easily spoofed by an attacker so the fingerprint is not useful in making security decisions. But the fingerprint is typically accurate enough to make policy decisions upon.

The fingerprints may be specified by operating system class, by version, or by subtype/patchlevel. The class of an operating system is typically the vendor or genre and would be OpenBSD for the pf(4) firewall itself. The version of the oldest available OpenBSD release on the main ftp site would be 2.6 and the fingerprint would be written

```
"OpenBSD 2.6"
```

The subtype of an operating system is typically used to describe the patchlevel if that patch led to changes in the TCP stack behavior. In the case of OpenBSD, the only subtype is for a fingerprint that was normalized by the *no-df* scrub option and would be specified as

```
"OpenBSD 3.3 no-df"
```

Fingerprints for most popular operating systems are provided by pf.os(5). Once pf(4) is running, a complete list of known operating system fingerprints may be listed by running:

```
# pfctl -so
```

Filter rules can enforce policy at any level of operating system specification assuming a fingerprint is present. Policy could limit traffic to approved operating systems or even ban traffic from hosts that aren't at the latest service pack.

The *unknown* class can also be used as the fingerprint which will match packets for which no operating system fingerprint is known.

Examples:

```
pass  out proto tcp from any os OpenBSD keep state
block out proto tcp from any os Doors
block out proto tcp from any os "Doors PT"
block out proto tcp from any os "Doors PT SP3"
block out from any os "unknown"
pass on lo0 proto tcp from any os "OpenBSD 3.3 lo0" keep state
```

Operating system fingerprinting is limited only to the TCP SYN packet. This means that it will not work on other protocols and will not match a currently established connection.

Caveat: operating system fingerprints are occasionally wrong. There are three problems: an attacker can trivially craft his packets to appear as any operating system he chooses; an operating system patch could change the stack behavior and no fingerprints will match it until the database is updated; and multiple operating systems may have the same fingerprint.

BLOCKING SPOOFED TRAFFIC

"Spoofing" is the faking of IP addresses, typically for malicious purposes. The *antispoof* directive expands to a set of filter rules which will block all traffic with a source IP from the network(s) directly connected to the specified interface(s) from entering the system through any other interface.

For example, the line

```
antispoof for lo0
```

expands to

```
block drop in on ! lo0 inet from 127.0.0.1/8 to any
block drop in on ! lo0 inet6 from ::1 to any
```

For non-loopback interfaces, there are additional rules to block incoming packets with a source IP address identical to the interface's IP(s). For example, assuming the interface wi0 had an IP address of 10.0.0.1 and a netmask of 255.255.255.0, the line

```
antispoof for wi0 inet
```

expands to

```
block drop in on ! wi0 inet from 10.0.0.0/24 to any
block drop in inet from 10.0.0.1 to any
```

Caveat: Rules created by the *antispoof* directive interfere with packets sent over loopback interfaces to local addresses. One should pass these explicitly.

FRAGMENT HANDLING

The size of IP datagrams (packets) can be significantly larger than the maximum transmission unit (MTU) of the network. In cases when it is necessary or more efficient to send such large packets, the large packet will be fragmented into many smaller packets that will each fit onto the wire. Unfortunately for a firewalling device, only the first logical fragment will contain the necessary header information for the subprotocol that allows pf(4) to filter on things such as TCP ports or to perform NAT.

Besides the use of *scrub* rules as described in **TRAFFIC NORMALIZATION** above, there are three options for handling fragments in the packet filter.

One alternative is to filter individual fragments with filter rules. If no *scrub* rule applies to a fragment, it is passed to the filter. Filter rules with matching IP header parameters decide whether the fragment is passed or blocked, in the same way as complete packets are filtered. Without reassembly, fragments can only be filtered based on IP header fields (source/destination address, protocol), since subprotocol header fields are not

available (TCP/UDP port numbers, ICMP code/type). The *fragment* option can be used to restrict filter rules to apply only to fragments, but not complete packets. Filter rules without the *fragment* option still apply to fragments, if they only specify IP header fields. For instance, the rule

```
pass in proto tcp from any to any port 80
```

never applies to a fragment, even if the fragment is part of a TCP packet with destination port 80, because without reassembly this information is not available for each fragment. This also means that fragments cannot create new or match existing state table entries, which makes stateful filtering and address translation (NAT, redirection) for fragments impossible.

It's also possible to reassemble only certain fragments by specifying source or destination addresses or protocols as parameters in *scrub* rules.

In most cases, the benefits of reassembly outweigh the additional memory cost, and it's recommended to use *scrub* rules to reassemble all fragments via the *fragment reassemble* modifier.

The memory allocated for fragment caching can be limited using pfctl(8). Once this limit is reached, fragments that would have to be cached are dropped until other entries time out. The timeout value can also be adjusted.

Currently, only IPv4 fragments are supported and IPv6 fragments are blocked unconditionally.

ANCHORS

Besides the main ruleset, pfctl(8) can load rulesets into *anchor* attachment points. An *anchor* is a container that can hold rules, address tables, and other anchors.

An *anchor* has a name which specifies the path where pfctl(8) can be used to access the anchor to perform operations on it, such as attaching child anchors to it or loading rules into it. Anchors may be nested, with components separated by '/' characters, similar to how file system hierarchies are laid out. The main ruleset is actually the default anchor, so filter and translation rules, for example, may also be contained in any anchor.

An anchor can reference another *anchor* attachment point using the following kinds of rules:

nat-anchor <name>
: Evaluates the *nat* rules in the specified *anchor*.

rdr-anchor <name>
: Evaluates the *rdr* rules in the specified *anchor*.

binat-anchor <name>
: Evaluates the *binat* rules in the specified *anchor*.

anchor <name>
: Evaluates the filter rules in the specified *anchor*.

load anchor <name> from <file>
: Loads the rules from the specified file into the anchor *name*.

When evaluation of the main ruleset reaches an *anchor* rule, pf(4) will proceed to evaluate all rules specified in that anchor.

Matching filter and translation rules in anchors with the *quick* option are final and abort the evaluation of the rules in other anchors and the main ruleset.

anchor rules are evaluated relative to the anchor in which they are contained. For example, all *anchor* rules specified in the main ruleset will reference anchor attachment points underneath the main ruleset, and *anchor* rules specified in a file loaded from a *load anchor* rule will be attached under that anchor point.

Rules may be contained in *anchor* attachment points which do not contain any rules when the main ruleset is loaded, and later such anchors can be manipulated through pfctl(8) without reloading the main ruleset or other anchors. For example,

```
ext_if = "kue0"
block on $ext_if all
anchor spam
pass out on $ext_if all keep state
pass in on $ext_if proto tcp from any \
      to $ext_if port smtp keep state
```

blocks all packets on the external interface by default, then evaluates all rules in the *anchor* named "spam", and finally passes all outgoing connections and incoming connections to port 25.

```
# echo "block in quick from 1.2.3.4 to any" | \
      pfctl -a spam -f -
```

This loads a single rule into the *anchor*, which blocks all packets from a specific address.

The anchor can also be populated by adding a *load anchor* rule after the *anchor* rule:

```
anchor spam
load anchor spam from "/etc/pf-spam.conf"
```

When pfctl(8) loads **pf.conf**, it will also load all the rules from the file /etc/pf-spam.conf into the anchor.

Optionally, *anchor* rules can specify the parameter's direction, interface, address family, protocol and source/destination address/port using the same syntax as filter rules. When parameters are used, the *anchor* rule is only evaluated for matching packets. This allows conditional evaluation of anchors, like:

```
block on $ext_if all
anchor spam proto tcp from any to any port smtp
pass out on $ext_if all keep state
pass in on $ext_if proto tcp from any to $ext_if port smtp keep state
```

The rules inside *anchor* spam are only evaluated for *tcp* packets with destination port 25. Hence,

```
# echo "block in quick from 1.2.3.4 to any" | \
      pfctl -a spam -f -
```

will only block connections from 1.2.3.4 to port 25.

Anchors may end with the asterisk ('*') character, which signifies that all anchors attached at that point should be evaluated in the alphabetical ordering of their anchor name. For example,

```
anchor "spam/*"
```

will evaluate each rule in each anchor attached to the spam anchor. Note that it will only evaluate anchors that are directly attached to the spam anchor, and will not descend to evaluate anchors recursively.

Since anchors are evaluated relative to the anchor in which they are contained, there is a mechanism for accessing the parent and ancestor anchors of a given anchor. Similar to file system path name resolution, if the sequence ".." appears as an anchor path component, the parent anchor of the current anchor in the path evaluation at that point will become the new current anchor. As an example, consider the following:

```
# echo ' anchor "spam/allowed" ' | pfctl -f -
# echo -e ' anchor "../banned" \n pass' | \
      pfctl -a spam/allowed -f -
```

Evaluation of the main ruleset will lead into the `spam/allowed` anchor, which will evaluate the rules in the `spam/banned` anchor, if any, before finally evaluating the *`pass`* rule.

Since the parser specification for anchor names is a string, any reference to an anchor name containing solidus (‘/’) characters will require double quote (‘"’) characters around the anchor name.

TRANSLATION EXAMPLES

This example maps incoming requests on port 80 to port 8080, on which a daemon is running (because, for example, it is not run as root, and therefore lacks permission to bind to port 80).

```
# use a macro for the interface name, so it can be changed easily
ext_if = "ne3"

# map daemon on 8080 to appear to be on 80
rdr on $ext_if proto tcp from any to any port 80 -> 127.0.0.1 port 8080
```

If the *`pass`* modifier is given, packets matching the translation rule are passed without inspecting the filter rules:

```
rdr pass on $ext_if proto tcp from any to any port 80 -> 127.0.0.1 \
      port 8080
```

In the example below, vlan12 is configured as 192.168.168.1; the machine translates all packets coming from 192.168.168.0/24 to 204.92.77.111 when they are going out any interface except vlan12. This has the net effect of making traffic from the 192.168.168.0/24 network appear as though it is the Internet routable address 204.92.77.111 to nodes behind any interface on the router except for the nodes on vlan12. (Thus, 192.168.168.1 can talk to the 192.168.168.0/24 nodes.)

```
nat on ! vlan12 from 192.168.168.0/24 to any -> 204.92.77.111
```

In the example below, the machine sits between a fake internal 144.19.74.* network, and a routable external IP of 204.92.77.100. The *`no nat`* rule excludes protocol AH from being translated.

```
# NO NAT
no nat on $ext_if proto ah from 144.19.74.0/24 to any
nat on $ext_if from 144.19.74.0/24 to any -> 204.92.77.100
```

In the example below, packets bound for one specific server, as well as those generated by the sysadmins are not proxied; all other connections are.

```
# NO RDR
no rdr on $int_if proto { tcp, udp } from any to $server port 80
no rdr on $int_if proto { tcp, udp } from $sysadmins to any port 80
rdr on $int_if proto { tcp, udp } from any to any port 80 -> 127.0.0.1 \
      port 80
```

This longer example uses both a NAT and a redirection. The external interface has the address 157.161.48.183. On the internal interface, we are running `ftp-proxy`(8), listening for outbound ftp sessions captured to port 8021.

```
# NAT
# Translate outgoing packets' source addresses (any protocol).
# In this case, any address but the gateway's external address is mapped.
nat on $ext_if inet from ! ($ext_if) to any -> ($ext_if)

# NAT PROXYING
# Map outgoing packets' source port to an assigned proxy port instead of
# an arbitrary port.
```

```
# In this case, proxy outgoing isakmp with port 500 on the gateway.
nat on $ext_if inet proto udp from any port = isakmp to any -> ($ext_if) \
      port 500

# BINAT
# Translate outgoing packets' source address (any protocol).
# Translate incoming packets' destination address to an internal machine
# (bidirectional).
binat on $ext_if from 10.1.2.150 to any -> $ext_if

# RDR
# Translate incoming packets' destination addresses.
# As an example, redirect a TCP and UDP port to an internal machine.
rdr on $ext_if inet proto tcp from any to ($ext_if) port 8080 \
      -> 10.1.2.151 port 22
rdr on $ext_if inet proto udp from any to ($ext_if) port 8080 \
      -> 10.1.2.151 port 53

# RDR
# Translate outgoing ftp control connections to send them to localhost
# for proxying with ftp-proxy(8) running on port 8021.
rdr on $int_if proto tcp from any to any port 21 -> 127.0.0.1 port 8021
```

In this example, a NAT gateway is set up to translate internal addresses using a pool of public addresses (192.0.2.16/28) and to redirect incoming web server connections to a group of web servers on the internal network.

```
# NAT LOAD BALANCE
# Translate outgoing packets' source addresses using an address pool.
# A given source address is always translated to the same pool address by
# using the source-hash keyword.
nat on $ext_if inet from any to any -> 192.0.2.16/28 source-hash

# RDR ROUND ROBIN
# Translate incoming web server connections to a group of web servers on
# the internal network.
rdr on $ext_if proto tcp from any to any port 80 \
      -> { 10.1.2.155, 10.1.2.160, 10.1.2.161 } round-robin
```

FILTER EXAMPLES

```
# The external interface is kue0
# (157.161.48.183, the only routable address)
# and the private network is 10.0.0.0/8, for which we are doing NAT.

# use a macro for the interface name, so it can be changed easily
ext_if = "kue0"

# normalize all incoming traffic
scrub in on $ext_if all fragment reassemble

# block and log everything by default
block return log on $ext_if all
```

```
# block anything coming from source we have no back routes for
block in from no-route to any

# block and log outgoing packets that do not have our address as source,
# they are either spoofed or something is misconfigured (NAT disabled,
# for instance), we want to be nice and do not send out garbage.
block out log quick on $ext_if from ! 157.161.48.183 to any

# silently drop broadcasts (cable modem noise)
block in quick on $ext_if from any to 255.255.255.255

# block and log incoming packets from reserved address space and invalid
# addresses, they are either spoofed or misconfigured, we cannot reply to
# them anyway (hence, no return-rst).
block in log quick on $ext_if from { 10.0.0.0/8, 172.16.0.0/12, \
      192.168.0.0/16, 255.255.255.255/32 } to any

# ICMP

# pass out/in certain ICMP queries and keep state (ping)
# state matching is done on host addresses and ICMP id (not type/code),
# so replies (like 0/0 for 8/0) will match queries
# ICMP error messages (which always refer to a TCP/UDP packet) are
# handled by the TCP/UDP states
pass on $ext_if inet proto icmp all icmp-type 8 code 0 keep state

# UDP

# pass out all UDP connections and keep state
pass out on $ext_if proto udp all keep state

# pass in certain UDP connections and keep state (DNS)
pass in on $ext_if proto udp from any to any port domain keep state

# TCP

# pass out all TCP connections and modulate state
pass out on $ext_if proto tcp all modulate state

# pass in certain TCP connections and keep state (SSH, SMTP, DNS, IDENT)
pass in on $ext_if proto tcp from any to any port { ssh, smtp, domain, \
      auth } flags S/SA keep state

# pass in data mode connections for ftp-proxy running on this host.
# (see ftp-proxy(8) for details)
pass in on $ext_if proto tcp from any to 157.161.48.183 port >= 49152 \
      flags S/SA keep state

# Do not allow Windows 9x SMTP connections since they are typically
# a viral worm. Alternately we could limit these OSes to 1 connection each.
block in on $ext_if proto tcp from any os {"Windows 95", "Windows 98"} \
      to any port smtp
```

```
# Packet Tagging

# three interfaces: $int_if, $ext_if, and $wifi_if (wireless). NAT is
# being done on $ext_if for all outgoing packets. tag packets in on
# $int_if and pass those tagged packets out on $ext_if.  all other
# outgoing packets (i.e., packets from the wireless network) are only
# permitted to access port 80.

pass in on $int_if from any to any tag INTNET keep state
pass in on $wifi_if from any to any keep state

block out on $ext_if from any to any
pass out quick on $ext_if tagged INTNET keep state
pass out on $ext_if proto tcp from any to any port 80 keep state

# tag incoming packets as they are redirected to spamd(8). use the tag
# to pass those packets through the packet filter.

rdr on $ext_if inet proto tcp from <spammers> to port smtp \
       tag SPAMD -> 127.0.0.1 port spamd

block in on $ext_if
pass in on $ext_if inet proto tcp tagged SPAMD keep state
```

GRAMMAR

Syntax for **pf.conf** in BNF:

```
line           = ( option | pf-rule | nat-rule | binat-rule | rdr-rule |
                 antispoof-rule | altq-rule | queue-rule | anchor-rule |
                 trans-anchors | load-anchors | table-rule )

option         = "set" ( [ "timeout" ( timeout | "{" timeout-list "}" ) ] |
                 [ "optimization" [ "default" | "normal" |
                 "high-latency" | "satellite" |
                 "aggressive" | "conservative" ] ]
                 [ "limit" ( limit-item | "{" limit-list "}" ) ] |
                 [ "loginterface" ( interface-name | "none" ) ] |
                 [ "block-policy" ( "drop" | "return" ) ] |
                 [ "state-policy" ( "if-bound" | "group-bound" |
                 "floating" ) ]
                 [ "require-order" ( "yes" | "no" ) ]
                 [ "fingerprints" filename ] |
                 [ "debug" ( "none" | "urgent" | "misc" | "loud" ) ] )

pf-rule        = action [ ( "in" | "out" ) ]
                 [ "log" | "log-all" ] [ "quick" ]
                 [ "on" ifspec ] [ route ] [ af ] [ protospec ]
                 hosts [ filteropt-list ]

filteropt-list = filteropt-list filteropt | filteropt
filteropt      = user | flags | icmp-type | icmp6-type | tos |
                 ( "keep" | "modulate" | "synproxy" ) "state"
```

```
                    [ "(" state-opts ")" ] |
                    "fragment" | "no-df" | "min-ttl" number |
                    "max-mss" number | "random-id" | "reassemble tcp" |
                    fragmentation | "allow-opts" |
                    "label" string | "tag" string | [ ! ] "tagged" string
                    "queue" ( string | "(" string [ [ "," ] string ] ")" ) |
                    "probability" number"%"

nat-rule       = [ "no" ] "nat" [ "pass" ] [ "on" ifspec ] [ af ]
                    [ protospec ] hosts [ "tag" string ] [ "tagged" string ]
                    [ "->" ( redirhost | "{" redirhost-list "}" )
                    [ portspec ] [ pooltype ] [ "static-port" ] ]
                    [ ( "if-bound" | "group-bound" | "floating" ) ]

binat-rule     = [ "no" ] "binat" [ "pass" ] [ "on" interface-name ]
                    [ af ] [ "proto" ( proto-name | proto-number ) ]
                    "from" address [ "/" mask-bits ] "to" ipspec
                    [ "tag" string ] [ "tagged" string ]
                    [ "->" address [ "/" mask-bits ] ]

rdr-rule       = [ "no" ] "rdr" [ "pass" ] [ "on" ifspec ] [ af ]
                    [ protospec ] hosts [ "tag" string ] [ "tagged" string ]
                    [ "->" ( redirhost | "{" redirhost-list "}" )
                    [ portspec ] [ pooltype ] ]
                    [ ( "if-bound" | "group-bound" | "floating" ) ]

antispoof-rule = "antispoof" [ "log" ] [ "quick" ]
                    "for" ( interface-name | "{" interface-list "}" )
                    [ af ] [ "label" string ]

table-rule     = "table" "<" string ">" [ tableopts-list ]
tableopts-list = tableopts-list tableopts | tableopts
tableopts      = "persist" | "const" | "file" string |
                    "{" [ tableaddr-list ] "}"
tableaddr-list = tableaddr-list [ "," ] tableaddr-spec | tableaddr-spec
tableaddr-spec = [ "!" ] tableaddr [ "/" mask-bits ]
tableaddr      = hostname | ipv4-dotted-quad | ipv6-coloned-hex |
                    interface-name | "self"

altq-rule      = "altq on" interface-name queueopts-list
                    "queue" subqueue
queue-rule     = "queue" string [ "on" interface-name ] queueopts-list
                    subqueue

anchor-rule    = "anchor" string [ ( "in" | "out" ) ] [ "on" ifspec ]
                    [ af ] [ "proto" ] [ protospec ] [ hosts ]

trans-anchors  = ( "nat-anchor" | "rdr-anchor" | "binat-anchor" ) string
                    [ "on" ifspec ] [ af ] [ "proto" ] [ protospec ] [ hosts ]

load-anchor    = "load anchor" string "from" filename
```

```
queueopts-list = queueopts-list queueopts | queueopts
queueopts      = [ "bandwidth" bandwidth-spec ] |
                 [ "qlimit" number ] | [ "tbrsize" number ] |
                 [ "priority" number ] | [ schedulers ]
schedulers     = ( cbq-def | priq-def | hfsc-def )
bandwidth-spec = "number" ( "b" | "Kb" | "Mb" | "Gb" | "%" )

action         = "pass" | "block" [ return ] | [ "no" ] "scrub"
return         = "drop" | "return" | "return-rst" [ "( ttl" number ")" ] |
                 "return-icmp" [ "(" icmpcode [ [ "," ] icmp6code ] ")" ] |
                 "return-icmp6" [ "(" icmp6code ")" ]
icmpcode       = ( icmp-code-name | icmp-code-number )
icmp6code      = ( icmp6-code-name | icmp6-code-number )

ifspec         = ( [ "!" ] interface-name ) | "{" interface-list "}"
interface-list = [ "!" ] interface-name [ [ "," ] interface-list ]
route          = "fastroute" |
                 ( "route-to" | "reply-to" | "dup-to" )
                 ( routehost | "{" routehost-list "}" )
                 [ pooltype ]
af             = "inet" | "inet6"

protospec      = "proto" ( proto-name | proto-number |
                 "{" proto-list "}" )
proto-list     = ( proto-name | proto-number ) [ [ "," ] proto-list ]

hosts          = "all" |
                 "from" ( "any" | "no-route" | "self" | host |
                 "{" host-list "}" | "route" string ) [ port ] [ os ]
                 "to"   ( "any" | "no-route" | "self" | host |
                 "{" host-list "}" | "route" string ) [ port ]

ipspec         = "any" | host | "{" host-list "}"
host           = [ "!" ] ( address [ "/" mask-bits ] | "<" string ">" )
redirhost      = address [ "/" mask-bits ]
routehost      = ( interface-name [ address [ "/" mask-bits ] ] )
address        = ( interface-name | "(" interface-name ")" | hostname |
                 ipv4-dotted-quad | ipv6-coloned-hex )
host-list      = host [ [ "," ] host-list ]
redirhost-list = redirhost [ [ "," ] redirhost-list ]
routehost-list = routehost [ [ "," ] routehost-list ]

port           = "port" ( unary-op | binary-op | "{" op-list "}" )
portspec       = "port" ( number | name ) [ ":" ( "*" | number | name ) ]
os             = "os"  ( os-name | "{" os-list "}" )
user           = "user" ( unary-op | binary-op | "{" op-list "}" )

unary-op       = [ "=" | "!=" | "<" | "<=" | ">" | ">=" ]
                 ( name | number )
binary-op      = number ( "<>" | "><" | ":" ) number
op-list        = ( unary-op | binary-op ) [ [ "," ] op-list ]
```

```
os-name        = operating-system-name
os-list        = os-name [ [ "," ] os-list ]

flags          = "flags" [ flag-set ] "/" flag-set
flag-set       = [ "F" ] [ "S" ] [ "R" ] [ "P" ] [ "A" ] [ "U" ] [ "E" ]
                 [ "W" ]

icmp-type      = "icmp-type" ( icmp-type-code | "{" icmp-list "}" )
icmp6-type     = "icmp6-type" ( icmp-type-code | "{" icmp-list "}" )
icmp-type-code = ( icmp-type-name | icmp-type-number )
                 [ "code" ( icmp-code-name | icmp-code-number ) ]
icmp-list      = icmp-type-code [ [ "," ] icmp-list ]

tos            = "tos" ( "lowdelay" | "throughput" | "reliability" |
                 [ "0x" ] number )

state-opts     = state-opt [ [ "," ] state-opts ]
state-opt      = ( "max" number | timeout |
                 "source-track" [ ( "rule" | "global" ) ] |
                 "max-src-nodes" number | "max-src-states" number |
                 "max-src-conn" number |
                 "max-src-conn-rate" number "/" number |
                 "overload" "<" string ">" [ "flush" ] |
                 "if-bound" | "group-bound" | "floating" )

fragmentation  = [ "fragment reassemble" | "fragment crop" |
                 "fragment drop-ovl" ]

timeout-list   = timeout [ [ "," ] timeout-list ]
timeout        = ( "tcp.first" | "tcp.opening" | "tcp.established" |
                 "tcp.closing" | "tcp.finwait" | "tcp.closed" |
                 "udp.first" | "udp.single" | "udp.multiple" |
                 "icmp.first" | "icmp.error" |
                 "other.first" | "other.single" | "other.multiple" |
                 "frag" | "interval" | "src.track" |
                 "adaptive.start" | "adaptive.end" ) number

limit-list     = limit-item [ [ "," ] limit-list ]
limit-item     = ( "states" | "frags" | "src-nodes" ) number

pooltype       = ( "bitmask" | "random" |
                 "source-hash" [ ( hex-key | string-key ) ] |
                 "round-robin" ) [ sticky-address ]

subqueue       = string | "{" queue-list "}"
queue-list     = string [ [ "," ] string ]
cbq-def        = "cbq" [ "(" cbq-opt [ [ "," ] cbq-opt ] ")" ]
priq-def       = "priq" [ "(" priq-opt [ [ "," ] priq-opt ] ")" ]
hfsc-def       = "hfsc" [ "(" hfsc-opt [ [ "," ] hfsc-opt ] ")" ]
cbq-opt        = ( "default" | "borrow" | "red" | "ecn" | "rio" )
priq-opt       = ( "default" | "red" | "ecn" | "rio" )
hfsc-opt       = ( "default" | "red" | "ecn" | "rio" |
```

```
                  linkshare-sc | realtime-sc | upperlimit-sc )
linkshare-sc   = "linkshare" sc-spec
realtime-sc    = "realtime" sc-spec
upperlimit-sc  = "upperlimit" sc-spec
sc-spec        = ( bandwidth-spec |
                  "(" bandwidth-spec number bandwidth-spec ")" )
```

FILES

`/etc/hosts`	Host name database.
`/etc/pf.conf`	Default location of the ruleset file.
`/etc/pf.os`	Default location of OS fingerprints.
`/etc/protocols`	Protocol name database.
`/etc/services`	Service name database.
`/usr/share/examples/pf`	Example rulesets.

SEE ALSO

`icmp`(4), `icmp6`(4), `ip`(4), `ip6`(4), `pf`(4), `route`(4), `tcp`(4), `udp`(4), `hosts`(5), `pf.os`(5), `protocols`(5), `services`(5), `ftp-proxy`(8), `pfctl`(8), `pflogd`(8), `route`(8)

HISTORY

The **`pf.conf`** file format first appeared in OpenBSD 3.0.

NAME

pf.os — format of the operating system fingerprints file

DESCRIPTION

The pf(4) firewall and the tcpdump(8) program can both fingerprint the operating system of hosts that originate an IPv4 TCP connection. The file consists of newline-separated records, one per fingerprint, containing nine colon (':') separated fields. These fields are as follows:

window	The TCP window size.
TTL	The IP time to live.
df	The presence of the IPv4 don't fragment bit.
packet size	The size of the initial TCP packet.
TCP options	An ordered list of the TCP options.
class	The class of operating system.
version	The version of the operating system.
subtype	The subtype of patchlevel of the operating system.
description	The overall textual description of the operating system, version and subtype.

The *window* field corresponds to the th->th_win field in the TCP header and is the source host's advertised TCP window size. It may be between zero and 65,535 inclusive. The window size may be given as a multiple of a constant by prepending the size with a percent sign '%' and the value will be used as a modulus. Three special values may be used for the window size:

*	An asterisk will wildcard the value so any window size will match.
S	Allow any window size which is a multiple of the maximum segment size (MSS).
T	Allow any window size which is a multiple of the maximum transmission unit (MTU).

The *ttl* value is the initial time to live in the IP header. The fingerprint code will account for the volatility of the packet's TTL as it traverses a network.

The *df* bit corresponds to the Don't Fragment bit in an IPv4 header. It tells intermediate routers not to fragment the packet and is used for path MTU discovery. It may be either a zero or a one.

The *packet size* is the literal size of the full IP packet and is a function of all of the IP and TCP options.

The *TCP options* field is an ordered list of the individual TCP options that appear in the SYN packet. Each option is described by a single character separated by a comma and certain ones may include a value. The options are:

Mnnn	maximum segment size (MSS) option. The value is the maximum packet size of the network link which may include the '%' modulus or match all MSSes with the '*' value.
N	the NOP option (NO Operation).
T[0]	the timestamp option. Certain operating systems always start with a zero timestamp in which case a zero value is added to the option; otherwise no value is appended.
S	the Selective ACKnowledgement OK (SACKOK) option.
Wnnn	window scaling option. The value is the size of the window scaling which may include the '%' modulus or match all window scalings with the '*' value.

No TCP options in the fingerprint may be given with a single dot '.'.

An example of OpenBSD's TCP options are:

```
M*,N,N,S,N,W0,N,N,T
```

The first option *M** is the MSS option and will match all values. The second and third options *N* will match two NOPs. The fourth option *S* will match the SACKOK option. The fifth *N* will match another NOP. The sixth *W0* will match a window scaling option with a zero scaling size. The seventh and eighth *N* options will

match two NOPs. And the ninth and final option *T* will match the timestamp option with any time value.

The TCP options in a fingerprint will only match packets with the exact same TCP options in the same order.

The *class* field is the class, genre or vendor of the operating system.

The *version* is the version of the operating system. It is used to distinguish between different fingerprints of operating systems of the same class but different versions.

The *subtype* is the subtype or patch level of the operating system version. It is used to distinguish between different fingerprints of operating systems of the same class and same version but slightly different patches or tweaking.

The *description* is a general description of the operating system, its version, patchlevel and any further useful details.

EXAMPLES

The fingerprint of a plain OpenBSD 3.3 host is:

```
16384:64:1:64:M*,N,N,S,N,W0,N,N,T:OpenBSD:3.3::OpenBSD 3.3
```

The fingerprint of an OpenBSD 3.3 host behind a PF scrubbing firewall with a no-df rule would be:

```
16384:64:0:64:M*,N,N,S,N,W0,N,N,T:OpenBSD:3.3:!df:OpenBSD 3.3 scrub no-df
```

An absolutely braindead embedded operating system fingerprint could be:

```
65535:255:0:40:.:DUMMY:1.1:p3:Dummy embedded OS v1.1p3
```

The tcpdump(8) output of

```
# tcpdump -s128 -c1 -nv 'tcp[13] == 2'
03:13:48.118526 10.0.0.1.3377 > 10.0.0.0.2: S [tcp sum ok] \
    534596083:534596083(0) win 57344 <mss 1460> (DF) [tos 0x10] \
    (ttl 64, id 11315)
```

almost translates into the following fingerprint

```
57344:64:1:44:M1460: exampleOS:1.0::exampleOS 1.0
```

tcpdump(8) does not explicitly give the packet length. But it can usually be derived by adding the size of the IPv4 header to the size of the TCP header to the size of the TCP options. The size of both headers is typically twenty each and the usual sizes of the TCP options are:

mss	four bytes.
nop	1 byte.
sackOK	two bytes.
timestamp	ten bytes.
wscale	three bytes.

In the above example, the packet size comes out to 44 bytes.

SEE ALSO

pf(4), pf.conf(5), pfctl(8), tcpdump(8)

NAME

pgsql_table − Postfix PostgreSQL client configuration

SYNOPSIS

postmap -q "*string***" pgsql:/etc/postfix/filename**

postmap -q - pgsql:/etc/postfix/*filename* <*inputfile*

DESCRIPTION

The Postfix mail system uses optional tables for address rewriting or mail routing. These tables are usually in **dbm** or **db** format.

Alternatively, lookup tables can be specified as PostgreSQL databases. In order to use PostgreSQL lookups, define a PostgreSQL source as a lookup table in main.cf, for example:

alias_maps = pgsql:/etc/pgsql-aliases.cf

The file /etc/postfix/pgsql-aliases.cf has the same format as the Postfix main.cf file, and can specify the parameters described below.

BACKWARDS COMPATIBILITY

For compatibility with other Postfix lookup tables, PostgreSQL parameters can also be defined in main.cf. In order to do that, specify as PostgreSQL source a name that doesn't begin with a slash or a dot. The PostgreSQL parameters will then be accessible as the name you've given the source in its definition, an underscore, and the name of the parameter. For example, if the map is specified as "pgsql:*pgsqlname*", the parameter "hosts" below would be defined in main.cf as "*pgsqlname*_hosts".

Note: with this form, the passwords for the PostgreSQL sources are written in main.cf, which is normally world-readable. Support for this form will be removed in a future Postfix version.

Postfix 2.2 has enhanced query interfaces for MySQL and PostgreSQL, these include features previously available only in the Postfix LDAP client. In the new interface the SQL query is specified via a single **query** parameter (described in more detail below). In Postfix 2.1 the parameter precedence was, from highest to lowest, **select_function**, **query** and finally **select_field**, ...

With Postfix 2.2 the **query** parameter has highest precedence, and is used in preference to the still supported, but slated to be phased out, **select_function**, **select_field**, **table**, **where_field** and **additional_conditions** parameters. To migrate to the new interface set:

query = SELECT *select_function*('%s')

or in the absence of **selection_function**, the lower precedence:

query = SELECT *select_field*
FROM *table*
WHERE *where_field* = '%s'
additional_conditions

Use the value, not the name, of each legacy parameter. Note that the **additional_conditions** parameter is optional and if not empty, will always start with **AND**.

LIST MEMBERSHIP

When using SQL to store lists such as $mynetworks, $mydestination, $relay_domains, $local_recipient_maps, etc., it is important to understand that the table must store each list member as a separate key. The table lookup verifies the *existence* of the key. See "Postfix lists versus tables" in the DATABASE_README document for a discussion.

Do NOT create tables that return the full list of domains in $mydestination or $relay_domains etc., or IP

addresses in $mynetworks.

DO create tables with each matching item as a key and with an arbitrary value. With SQL databases it is not uncommon to return the key itself or a constant value.

PGSQL PARAMETERS

hosts The hosts that Postfix will try to connect to and query from. Specify *unix:* for UNIX-domain sockets, *inet:* for TCP connections (default). Example:

hosts = host1.some.domain host2.some.domain
hosts = unix:/file/name

The hosts are tried in random order, with all connections over UNIX domain sockets being tried before those over TCP. The connections are automatically closed after being idle for about 1 minute, and are re-opened as necessary.

NOTE: the *unix:* and *inet:* prefixes are accepted for backwards compatibility reasons, but are actually ignored. The PostgreSQL client library will always try to connect to an UNIX socket if the name starts with a slash, and will try a TCP connection otherwise.

user, password

The user name and password to log into the pgsql server. Example:

user = someone
password = some_password

dbname

The database name on the servers. Example:

dbname = customer_database

query The SQL query template used to search the database, where **%s** is a substitute for the address Postfix is trying to resolve, e.g.

query = SELECT replacement FROM aliases WHERE mailbox = '%s'

This parameter supports the following '%' expansions:

%% This is replaced by a literal '%' character. (Postfix 2.2 and later)

%s This is replaced by the input key. SQL quoting is used to make sure that the input key does not add unexpected metacharacters.

%u When the input key is an address of the form user@domain, **%u** is replaced by the SQL quoted local part of the address. Otherwise, **%u** is replaced by the entire search string. If the localpart is empty, the query is suppressed and returns no results.

%d When the input key is an address of the form user@domain, **%d** is replaced by the SQL quoted domain part of the address. Otherwise, the query is suppressed and returns no results.

%[SUD]

The upper-case equivalents of the above expansions behave in the **query** parameter identically to their lower-case counter-parts. With the **result_format** parameter (see below), they expand the input key rather than the result value.

The above %S, %U and %D expansions are available with Postfix 2.2 and later

%[1-9] The patterns %1, %2, ... %9 are replaced by the corresponding most significant component of the input key's domain. If the input key is *user@mail.example.com*, then %1 is **com**, %2 is **example** and %3 is **mail**. If the input key is unqualified or does not have enough domain components to satisfy all the specified patterns, the query is suppressed and returns no results.

The above %1, ... %9 expansions are available with Postfix 2.2 and later

The **domain** parameter described below limits the input keys to addresses in matching domains. When the **domain** parameter is non-empty, SQL queries for unqualified addresses or addresses in non-matching domains are suppressed and return no results.

The precedence of this parameter has changed with Postfix 2.2, in prior releases the precedence was, from highest to lowest, **select_function**, **query**, **select_field**, ...

With Postfix 2.2 the **query** parameter has highest precedence, see COMPATIBILITY above.

NOTE: DO NOT put quotes around the **query** parameter.

result_format (default: %s)

Format template applied to result attributes. Most commonly used to append (or prepend) text to the result. This parameter supports the following '%' expansions:

%% This is replaced by a literal '%' character.

%s This is replaced by the value of the result attribute. When result is empty it is skipped.

%u When the result attribute value is an address of the form user@domain, **%u** is replaced by the local part of the address. When the result has an empty localpart it is skipped.

%d When a result attribute value is an address of the form user@domain, **%d** is replaced by the domain part of the attribute value. When the result is unqualified it is skipped.

%[SUD1-9]

The upper-case and decimal digit expansions interpolate the parts of the input key rather than the result. Their behavior is identical to that described with **query**, and in fact because the input key is known in advance, queries whose key does not contain all the information specified in the result template are suppressed and return no results.

For example, using "result_format = smtp:[%s]" allows one to use a mailHost attribute as the basis of a transport(5) table. After applying the result format, multiple values are concatenated as comma separated strings. The expansion_limit and parameter explained below allows one to restrict the number of values in the result, which is especially useful for maps that must return at most one value.

The default value **%s** specifies that each result value should be used as is.

This parameter is available with Postfix 2.2 and later.

NOTE: DO NOT put quotes around the result format!

domain (default: no domain list)

This is a list of domain names, paths to files, or dictionaries. When specified, only fully qualified search keys with a *non-empty* localpart and a matching domain are eligible for lookup: 'user' lookups, bare domain lookups and "@domain" lookups are not performed. This can significantly reduce the query load on the PostgreSQL server.

domain = postfix.org, hash:/etc/postfix/searchdomains

It is best not to use SQL to store the domains eligible for SQL lookups.

This parameter is available with Postfix 2.2 and later.

NOTE: DO NOT define this parameter for local(8) aliases, because the input keys are always unqualified.

expansion_limit (default: 0)

A limit on the total number of result elements returned (as a comma separated list) by a lookup against the map. A setting of zero disables the limit. Lookups fail with a temporary error if the

limit is exceeded. Setting the limit to 1 ensures that lookups do not return multiple values.

Pre-Postfix 2.2 legacy interfaces:

select_function

This parameter specifies a database function name. Example:

select_function = my_lookup_user_alias

This is equivalent to:

query = SELECT my_lookup_user_alias('%s')

This parameter overrides the legacy table-related fields (described below). With Postfix versions prior to 2.2, it also overrides the **query** parameter. Starting with Postfix 2.2, the **query** parameter has highest precedence, and this parameter is deprecated. Please migrate to the new **query** interface as this interface is slated to be phased out.

The following parameters (with lower precedence than the **select_function** interface described above) can be used to build the SQL select statement as follows:

SELECT [**select_field**]
FROM [**table**]
WHERE [**where_field**] = '%s'
[**additional_conditions**]

The specifier %s is replaced with each lookup by the lookup key and is escaped so if it contains single quotes or other odd characters, it will not cause a parse error, or worse, a security problem.

Starting with Postfix 2.2, this interface is obsoleted by the more general **query** interface described above. If higher precedence the **query** or **select_function** parameters described above are defined, these parameters are ignored. Please migrate to the new **query** interface as this interface is slated to be phased out.

select_field

The SQL "select" parameter. Example:

select_field = forw_addr

table The SQL "select .. from" table name. Example:

table = mxaliases

where_field

The SQL "select .. where" parameter. Example:

where_field = alias

additional_conditions

Additional conditions to the SQL query. Example:

additional_conditions = AND status = 'paid'

SEE ALSO

postmap(1), Postfix lookup table manager
postconf(5), configuration parameters
ldap_table(5), LDAP lookup tables
mysql_table(5), MySQL lookup tables

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview
PGSQL_README, Postfix PostgreSQL client guide

LICENSE

The Secure Mailer license must be distributed with this software.

HISTORY

PgSQL support was introduced with Postfix version 2.1.

AUTHOR(S)

Based on the MySQL client by:
Scott Cotton, Joshua Marcus
IC Group, Inc.

Ported to PostgreSQL by:
Aaron Sethman

Further enhanced by:
Liviu Daia
Institute of Mathematics of the Romanian Academy
P.O. BOX 1-764
RO-014700 Bucharest, ROMANIA

NAME

phones — remote host phone number data base

DESCRIPTION

The file `/etc/phones` contains the system-wide private phone numbers for the `tip`(1) program. This file is normally unreadable, and so may contain privileged information.

The format of the file is a series of lines containing whitespace separate fields, of the form:

```
system-name phone-number
```

The *system-name* is one of those defined in the `remote`(5) file.

The *phone-number* is constructed from any sequence of characters terminated only by a comma (“,”) or the end of the line. The equals (“=”) and asterisk (“∗”) characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The “=” DF02-AC and the “∗” is required by the BIZCOMP 1030.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name `tip`(1) will attempt to dial each one in turn, until it establishes a connection.

FILES

`/etc/phones`

SEE ALSO

`tip`(1), `remote`(5)

HISTORY

The **phones** file appeared in 4.2BSD.

NAME

pkg_summary — summary of binary package repository

DESCRIPTION

The file **pkg_summary** contains information about each package in a binary package repository as a list of variable-value pairs. The variables describing different packages are separated by one empty line. Each line has the format VARIABLE=VALUE. If the value consists of more than one line, each line is prefixed with VARIABLE=. Multi-line variables are guaranteed to be in consecutive lines.

The following variables are used:

BUILD_DATE
: (required) The date and time when the package was built.

CATEGORIES
: (required) A list of categories which this package fits in, separated by space.

COMMENT
: (required) A one-line description of the package.

CONFLICTS
: (optional) A list of dewey patterns of packages the package conflicts with, one per line. If missing, this package has no conflicts.

DEPENDS
: (optional) A list of dewey patterns of packages the package depends on, one per line. If missing, this package has no dependencies.

DESCRIPTION
: (required) A more detailed description of the package.

FILE_NAME
: (optional) The name of the binary package file. If not given, PKGNAME.tgz can be assumed.

FILE_SIZE
: (optional) The size of the binary package file, in bytes.

HOMEPAGE
: (optional) A URL where more information about the package can be found.

LICENSE
: (optional) The type of license this package is distributed under. If empty or missing, it is OSI-approved.

MACHINE_ARCH
: (required) The architecture on which the package was compiled.

OPSYS (required) The operating system on which the package was compiled.

OS_VERSION
: (required) The version of the operating system on which the package was compiled.

PKG_OPTIONS
: (optional) Any options selected to compile this package. If missing, the package does not support options.

PKGNAME
: (required) The name of the package.

`PKGPATH`
(required) The path of the package directory within pkgsrc.

`PKGTOOLS_VERSION`
(required) The version of the package tools used to create the package.

`PROVIDES`
(optional) A list of shared libraries provided by the package, including major version number, one per line. If missing, this package does not provide shared libraries.

`REQUIRES`
(optional) A list of shared libraries needed by the package, including major version number, one per line. If missing, this package does not require shared libraries.

`SIZE_PKG`
(required) The size of the package when installed, in bytes.

The **`pkg_summary`** file can be generated using the `pkg_info`(1) **`-X`** option. For example, the following will list this data for all installed packages:

```
pkg_info -X -a
```

SEE ALSO

`pkg_info`(1)

HISTORY

The **`pkg_summary`** format was first officially documented in April 2006.

NAME

postconf − Postfix configuration parameters

SYNOPSIS

postconf *parameter ...*

postconf -e "*parameter=value*" ...

DESCRIPTION

The Postfix main.cf configuration file specifies a small subset of all the parameters that control the operation of the Postfix mail system. Parameters not specified in main.cf are left at their default values.

The general format of the main.cf file is as follows:

- Each logical line has the form "parameter = value". Whitespace around the "=" is ignored, as is whitespace at the end of a logical line.
- Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.
- A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.
- A parameter value may refer to other parameters.
  - The expressions "$name", "${name}" or "$(name)" are recursively replaced by the value of the named parameter.
  - The expression "${name?value}" expands to "value" when "$name" is non-empty. This form is supported with Postfix version 2.2 and later.
  - The expression "${name:value}" expands to "value" when "$name" is empty. This form is supported with Postfix version 2.2 and later.
  - Specify "$$" to produce a single "$" character.
- When the same parameter is defined multiple times, only the last instance is remembered.
- Otherwise, the order of main.cf parameter definitions does not matter.

The remainder of this document is a description of all Postfix configuration parameters. Default values are shown after the parameter name in parentheses, and can be looked up with the "**postconf -d**" command.

Note: this is not an invitation to make changes to Postfix configuration parameters. Unnecessary changes can impair the operation of the mail system.

2bounce_notice_recipient (default: postmaster)

The recipient of undeliverable mail that cannot be returned to the sender. This feature is enabled with the notify_classes parameter.

access_map_reject_code (default: 554)

The numerical Postfix SMTP server response code when a client is rejected by an **access**(5) map restriction.

Do not change this unless you have a complete understanding of RFC 821.

address_verify_default_transport (default: $default_transport)

Overrides the default_transport parameter setting for address verification probes.

This feature is available in Postfix 2.1 and later.

address_verify_local_transport (default: $local_transport)

Overrides the local_transport parameter setting for address verification probes.

This feature is available in Postfix 2.1 and later.

address_verify_map (default: empty)

Optional lookup table for persistent address verification status storage. The table is maintained by the **verify**(8) service, and is opened before the process releases privileges.

By default, the information is kept in volatile memory, and is lost after "**postfix reload**" or "**postfix stop**".

Specify a location in a file system that will not fill up. If the database becomes corrupted, the world comes to an end. To recover delete the file and do "**postfix reload**".

Examples:

```
address_verify_map = hash:/etc/postfix/verify
address_verify_map = btree:/etc/postfix/verify
```

This feature is available in Postfix 2.1 and later.

address_verify_negative_cache (default: yes)
Enable caching of failed address verification probe results. When this feature is enabled, the cache may pollute quickly with garbage. When this feature is disabled, Postfix will generate an address probe for every lookup.

This feature is available in Postfix 2.1 and later.

address_verify_negative_expire_time (default: 3d)
The time after which a failed probe expires from the address verification cache.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks).

This feature is available in Postfix 2.1 and later.

address_verify_negative_refresh_time (default: 3h)
The time after which a failed address verification probe needs to be refreshed.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks).

This feature is available in Postfix 2.1 and later.

address_verify_poll_count (default: 3)
How many times to query the **verify**(8) service for the completion of an address verification request in progress.

The default poll count is 3.

Specify 1 to implement a crude form of greylisting, that is, always defer the first delivery request for a never seen before address.

Example:

```
address_verify_poll_count = 1
```

This feature is available in Postfix 2.1 and later.

address_verify_poll_delay (default: 3s)
The delay between queries for the completion of an address verification request in progress.

The default polling delay is 3 seconds.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks).

This feature is available in Postfix 2.1 and later.

address_verify_positive_expire_time (default: 31d)
The time after which a successful probe expires from the address verification cache.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks).

This feature is available in Postfix 2.1 and later.

address_verify_positive_refresh_time (default: 7d)
The time after which a successful address verification probe needs to be refreshed. The address verification status is not updated when the probe fails (optimistic caching).

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks).

This feature is available in Postfix 2.1 and later.

address_verify_relay_transport (default: $relay_transport)
Overrides the relay_transport parameter setting for address verification probes.

This feature is available in Postfix 2.1 and later.

address_verify_relayhost (default: $relayhost)
Overrides the relayhost parameter setting for address verification probes. This information can be overruled with the **transport**(5) table.

This feature is available in Postfix 2.1 and later.

address_verify_sender (default: postmaster)
The sender address to use in address verification probes. To avoid problems with address probes that are sent in response to address probes, the Postfix SMTP server excludes the probe sender address from all SMTPD access blocks.

Specify an empty value (address_verify_sender =) or <> if you want to use the null sender address. Beware, some sites reject mail from <>, even though RFCs require that such addresses be accepted.

Examples:

```
address_verify_sender = <>
address_verify_sender = postmaster@my.domain
```

This feature is available in Postfix 2.1 and later.

address_verify_sender_dependent_relayhost_maps (default: empty)
Overrides the sender_dependent_relayhost_maps parameter setting for address verification probes.

This feature is available in Postfix 2.3 and later.

address_verify_service_name (default: verify)
The name of the **verify**(8) address verification service. This service maintains the status of sender and/or recipient address verification probes, and generates probes on request by other Postfix processes.

address_verify_transport_maps (default: $transport_maps)
Overrides the transport_maps parameter setting for address verification probes.

This feature is available in Postfix 2.1 and later.

address_verify_virtual_transport (default: $virtual_transport)
Overrides the virtual_transport parameter setting for address verification probes.

This feature is available in Postfix 2.1 and later.

alias_database (default: see postconf -d output)
The alias databases for **local**(8) delivery that are updated with "**newaliases**" or with "**sendmail -bi**".

This is a separate configuration parameter because not all the tables specified with $alias_maps have to be local files.

Examples:

```
alias_database = hash:/etc/aliases
alias_database = hash:/etc/mail/aliases
```

alias_maps (default: see postconf -d output)
The alias databases that are used for **local**(8) delivery. See **aliases**(5) for syntax details.

The default list is system dependent. On systems with NIS, the default is to search the local alias database, then the NIS alias database.

If you change the alias database, run "**postalias /etc/aliases**" (or wherever your system stores the mail alias file), or simply run "**newaliases**" to build the necessary DBM or DB file.

The **local**(8) delivery agent disallows regular expression substitution of $1 etc. in alias_maps, because that would open a security hole.

The **local**(8) delivery agent will silently ignore requests to use the **proxymap**(8) server within alias_maps.

Instead it will open the table directly. Before Postfix version 2.2, the **local**(8) delivery agent will terminate with a fatal error.

Examples:

```
alias_maps = hash:/etc/aliases, nis:mail.aliases
alias_maps = hash:/etc/aliases
```

allow_mail_to_commands (default: alias, forward)

Restrict **local**(8) mail delivery to external commands. The default is to disallow delivery to "|command" in :include: files (see **aliases**(5) for the text that defines this terminology).

Specify zero or more of: **alias**, **forward** or **include**, in order to allow commands in **aliases**(5), .forward files or in :include: files, respectively.

Example:

```
allow_mail_to_commands = alias,forward,include
```

allow_mail_to_files (default: alias, forward)

Restrict **local**(8) mail delivery to external files. The default is to disallow "/file/name" destinations in :include: files (see **aliases**(5) for the text that defines this terminology).

Specify zero or more of: **alias**, **forward** or **include**, in order to allow "/file/name" destinations in **aliases**(5), .forward files and in :include: files, respectively.

Example:

```
allow_mail_to_files = alias,forward,include
```

allow_min_user (default: no)

Allow a recipient address to have '-' as the first character. By default, this is not allowed, to avoid accidents with software that passes email addresses via the command line. Such software would not be able to distinguish a malicious address from a bona fide command-line option. Although this can be prevented by inserting a "--" option terminator into the command line, this is difficult to enforce consistently and globally.

allow_percent_hack (default: yes)

Enable the rewriting of the form "user%domain" to "user@domain". This is enabled by default.

Note: with Postfix version 2.2, message header address rewriting happens only when one of the following conditions is true:

- The message is received with the Postfix **sendmail**(1) command,
- The message is received from a network client that matches $local_header_rewrite_clients,
- The message is received from the network, and the remote_header_rewrite_domain parameter specifies a non-empty value.

To get the behavior before Postfix version 2.2, specify "local_header_rewrite_clients = static:all".

Example:

```
allow_percent_hack = no
```

allow_untrusted_routing (default: no)

Forward mail with sender-specified routing (user[@%!]remote[@%!]site) from untrusted clients to destinations matching $relay_domains.

By default, this feature is turned off. This closes a nasty open relay loophole where a backup MX host can be tricked into forwarding junk mail to a primary MX host which then spams it out to the world.

This parameter also controls if non-local addresses with sender-specified routing can match Postfix access tables. By default, such addresses cannot match Postfix access tables, because the address is ambiguous.

alternate_config_directories (default: empty)

A list of non-default Postfix configuration directories that may be specified with "-c config_directory" on the command line, or via the MAIL_CONFIG environment parameter.

This list must be specified in the default Postfix configuration directory, and is used by set-gid Postfix commands such as **postqueue**(1) and **postdrop**(1).

always_bcc (default: empty)

Optional address that receives a "blind carbon copy" of each message that is received by the Postfix mail system.

Note: if mail to the BCC address bounces it will be returned to the sender.

Note: automatic BCC recipients are produced only for new mail. To avoid mailer loops, automatic BCC recipients are not generated for mail that Postfix forwards internally, nor for mail that Postfix generates itself.

anvil_rate_time_unit (default: 60s)

The time unit over which client connection rates and other rates are calculated.

This feature is implemented by the **anvil**(8) service which is available in Postfix version 2.2 and later.

The default interval is relatively short. Because of the high frequency of updates, the **anvil**(8) server uses volatile memory only. Thus, information is lost whenever the process terminates.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

anvil_status_update_time (default: 600s)

How frequently the **anvil**(8) connection and rate limiting server logs peak usage information.

This feature is available in Postfix 2.2 and later.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

append_at_myorigin (default: yes)

With locally submitted mail, append the string "@$myorigin" to mail addresses without domain information. With remotely submitted mail, append the string "@$remote_header_rewrite_domain" instead.

Note 1: this feature is enabled by default and must not be turned off. Postfix does not support domain-less addresses.

Note 2: with Postfix version 2.2, message header address rewriting happens only when one of the following conditions is true:

- The message is received with the Postfix **sendmail**(1) command,
- The message is received from a network client that matches $local_header_rewrite_clients,
- The message is received from the network, and the remote_header_rewrite_domain parameter specifies a non-empty value.

To get the behavior before Postfix version 2.2, specify "local_header_rewrite_clients = static:all".

append_dot_mydomain (default: yes)

With locally submitted mail, append the string ".$mydomain" to addresses that have no ".domain" information. With remotely submitted mail, append the string ".$remote_header_rewrite_domain" instead.

Note 1: this feature is enabled by default. If disabled, users will not be able to send mail to "user@partial-domainname" but will have to specify full domain names instead.

Note 2: with Postfix version 2.2, message header address rewriting happens only when one of the following conditions is true:

- The message is received with the Postfix **sendmail**(1) command,
- The message is received from a network client that matches $local_header_rewrite_clients,
- The message is received from the network, and the remote_header_rewrite_domain parameter specifies a non-empty value.

To get the behavior before Postfix version 2.2, specify "local_header_rewrite_clients = static:all".

application_event_drain_time (default: 100s)
How long the **postkick**(1) command waits for a request to enter the server's input buffer before giving up.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.1 and later.

authorized_flush_users (default: static:anyone)
List of users who are authorized to flush the queue.

By default, all users are allowed to flush the queue. Access is always granted if the invoking user is the super-user or the $mail_owner user. Otherwise, the real UID of the process is looked up in the system password file, and access is granted only if the corresponding login name is on the access list. The username "unknown" is used for processes whose real UID is not found in the password file.

Specify a list of user names, "/file/name" or "type:table" patterns, separated by commas and/or whitespace. The list is matched left to right, and the search stops on the first match. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a name matches a lookup key (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude a name from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

This feature is available in Postfix 2.2 and later.

authorized_mailq_users (default: static:anyone)
List of users who are authorized to view the queue.

By default, all users are allowed to view the queue. Access is always granted if the invoking user is the super-user or the $mail_owner user. Otherwise, the real UID of the process is looked up in the system password file, and access is granted only if the corresponding login name is on the access list. The username "unknown" is used for processes whose real UID is not found in the password file.

Specify a list of user names, "/file/name" or "type:table" patterns, separated by commas and/or whitespace. The list is matched left to right, and the search stops on the first match. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a name matches a lookup key (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude a user name from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

This feature is available in Postfix 2.2 and later.

authorized_submit_users (default: static:anyone)
List of users who are authorized to submit mail with the **sendmail**(1) command (and with the privileged **postdrop**(1) helper command).

By default, all users are allowed to submit mail. Otherwise, the real UID of the process is looked up in the system password file, and access is granted only if the corresponding login name is on the access list. The username "unknown" is used for processes whose real UID is not found in the password file. To deny mail submission access to all users specify an empty list.

Specify a list of user names, "/file/name" or "type:table" patterns, separated by commas and/or whitespace. The list is matched left to right, and the search stops on the first match. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a name matches a lookup key (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude a user name from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Example:

```
authorized_submit_users = !www, static:all
```

This feature is available in Postfix 2.2 and later.

authorized_verp_clients (default: $mynetworks)
What SMTP clients are allowed to specify the XVERP command. This command requests that mail be delivered one recipient at a time with a per recipient return address.

By default, only trusted clients are allowed to specify XVERP.

This parameter was introduced with Postfix version 1.1. Postfix version 2.1 renamed this parameter to smtpd_authorized_verp_clients and changed the default to none.

Specify a list of network/netmask patterns, separated by commas and/or whitespace. The mask specifies the number of bits in the network part of a host address. You can also specify hostnames or \&.domain names (the initial dot causes the domain to match any name below it), "/file/name" or "type:table" patterns. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a table entry matches a lookup string (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude an address or network block from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Note: IP version 6 address information must be specified inside [] in the authorized_verp_clients value, and in files specified with "/file/name". IP version 6 addresses contain the ":" character, and would otherwise be confused with a "type:table" pattern.

backwards_bounce_logfile_compatibility (default: yes)

Produce additional **bounce**(8) logfile records that can be read by Postfix versions before 2.0. The current and more extensible "name = value" format is needed in order to implement more sophisticated functionality.

This feature is available in Postfix 2.1 and later.

berkeley_db_create_buffer_size (default: 16777216)

The per-table I/O buffer size for programs that create Berkeley DB hash or btree tables. Specify a byte count.

This feature is available in Postfix 2.0 and later.

berkeley_db_read_buffer_size (default: 131072)

The per-table I/O buffer size for programs that read Berkeley DB hash or btree tables. Specify a byte count.

This feature is available in Postfix 2.0 and later.

best_mx_transport (default: empty)

Where the Postfix SMTP client should deliver mail when it detects a "mail loops back to myself" error condition. This happens when the local MTA is the best SMTP mail exchanger for a destination not listed in $mydestination, $inet_interfaces, $proxy_interfaces, $virtual_alias_domains, or $virtual_mailbox_domains. By default, the Postfix SMTP client returns such mail as undeliverable.

Specify, for example, "best_mx_transport = local" to pass the mail from the Postfix SMTP client to the **local**(8) delivery agent. You can specify any message delivery "transport" or "transport:nexthop" that is defined in the master.cf file. See the **transport**(5) manual page for the syntax and meaning of "transport" or "transport:nexthop".

However, this feature is expensive because it ties up a Postfix SMTP client process while the **local**(8) delivery agent is doing its work. It is more efficient (for Postfix) to list all hosted domains in a table or database.

biff (default: yes)

Whether or not to use the local biff service. This service sends "new mail" notifications to users who have requested new mail notification with the UNIX command "biff y".

For compatibility reasons this feature is on by default. On systems with lots of interactive users, the biff service can be a performance drain. Specify "biff = no" in main.cf to disable.

body_checks (default: empty)

Optional lookup tables for content inspection as specified in the **body_checks**(5) manual page.

Note: with Postfix versions before 2.0, these rules inspect all content after the primary message headers.

body_checks_size_limit (default: 51200)

How much text in a message body segment (or attachment, if you prefer to use that term) is subjected to body_checks inspection. The amount of text is limited to avoid scanning huge attachments.

This feature is available in Postfix 2.0 and later.

bounce_notice_recipient (default: postmaster)
The recipient of postmaster notifications with the message headers of mail that Postfix did not deliver and of SMTP conversation transcripts of mail that Postfix did not receive. This feature is enabled with the notify_classes parameter.

bounce_queue_lifetime (default: 5d)
The maximal time a bounce message is queued before it is considered undeliverable. By default, this is the same as the queue life time for regular mail.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is d (days).

Specify 0 when mail delivery should be tried only once.

This feature is available in Postfix 2.1 and later.

bounce_service_name (default: bounce)
The name of the **bounce**(8) service. This service maintains a record of failed delivery attempts and generates non-delivery notifications.

This feature is available in Postfix 2.0 and later.

bounce_size_limit (default: 50000)
The maximal amount of original message text that is sent in a non-delivery notification. Specify a byte count. If you increase this limit, then you should increase the mime_nesting_limit value proportionally.

bounce_template_file (default: empty)
Pathname of a configuration file with bounce message templates. These override the built-in templates of delivery status notification (DSN) messages for undeliverable mail, for delayed mail, successful delivery, or delivery verification. The **bounce**(5) manual page describes how to edit and test template files.

Template message body text may contain $name references to Postfix configuration parameters. The result of $name expansion can be previewed with "**postconf -b** *file_name*" before the file is placed into the Postfix configuration directory.

This feature is available in Postfix 2.3 and later.

broken_sasl_auth_clients (default: no)
Enable inter-operability with SMTP clients that implement an obsolete version of the AUTH command (RFC 2554). Examples of such clients are MicroSoft Outlook Express version 4 and MicroSoft Exchange version 5.0.

Specify "broken_sasl_auth_clients = yes" to have Postfix advertise AUTH support in a non-standard way.

canonical_classes (default: envelope_sender, envelope_recipient, header_sender, header_recipient)
What addresses are subject to canonical_maps address mapping. By default, canonical_maps address mapping is applied to envelope sender and recipient addresses, and to header sender and header recipient addresses.

Specify one or more of: envelope_sender, envelope_recipient, header_sender, header_recipient

This feature is available in Postfix 2.2 and later.

canonical_maps (default: empty)
Optional address mapping lookup tables for message headers and envelopes. The mapping is applied to both sender and recipient addresses, in both envelopes and in headers, as controlled with the canonical_classes parameter. This is typically used to clean up dirty addresses from legacy mail systems, or to replace login names by Firstname.Lastname. The table format and lookups are documented in **canonical**(5). For an overview of Postfix address manipulations see the ADDRESS_REWRITING_README document.

If you use this feature, run "**postmap /etc/postfix/canonical**" to build the necessary DBM or DB file after every change. The changes will become visible after a minute or so. Use "**postfix reload**" to eliminate the delay.

Note: with Postfix version 2.2, message header address mapping happens only when message header

address rewriting is enabled:

- The message is received with the Postfix **sendmail**(1) command,
- The message is received from a network client that matches $local_header_rewrite_clients,
- The message is received from the network, and the remote_header_rewrite_domain parameter specifies a non-empty value.

To get the behavior before Postfix version 2.2, specify "local_header_rewrite_clients = static:all".

Examples:

```
canonical_maps = dbm:/etc/postfix/canonical
canonical_maps = hash:/etc/postfix/canonical
```

cleanup_service_name (default: cleanup)

The name of the **cleanup**(8) service. This service rewrites addresses into the standard form, and performs **canonical**(5) address mapping and **virtual**(5) aliasing.

This feature is available in Postfix 2.0 and later.

command_directory (default: see postconf -d output)

The location of all postfix administrative commands.

command_execution_directory (default: empty)

The **local**(8) delivery agent working directory for delivery to external command. Failure to change directory causes the delivery to be deferred.

The following $name expansions are done on command_execution_directory before the directory is changed. Expansion happens in the context of the delivery request. The result of $name expansion is filtered with the character set that is specified with the execution_directory_expansion_filter parameter.

$user The recipient's username.

$shell The recipient's login shell pathname.

$home The recipient's home directory.

$recipient
The full recipient address.

$extension
The optional recipient address extension.

$domain
The recipient domain.

$local The entire recipient localpart.

$recipient_delimiter
The system-wide recipient address extension delimiter.

${name?value}
Expands to *value* when *$name* is non-empty.

${name:value}
Expands to *value* when *$name* is empty.

Instead of $name you can also specify ${name} or $(name).

This feature is available in Postfix 2.2 and later.

command_expansion_filter (default: see postconf -d output)

Restrict the characters that the **local**(8) delivery agent allows in $name expansions of $mailbox_command. Characters outside the allowed set are replaced by underscores.

command_time_limit (default: 1000s)
Time limit for delivery to external commands. This limit is used by the **local**(8) delivery agent, and is the default time limit for delivery by the **pipe**(8) delivery agent.

Note: if you set this time limit to a large value you must update the global ipc_timeout parameter as well.

config_directory (default: see postconf -d output)
The default location of the Postfix main.cf and master.cf configuration files. This can be overruled via the following mechanisms:

- The MAIL_CONFIG environment variable (daemon processes and commands).
- The "-c" command-line option (commands only).

With Postfix command that run with set-gid privileges, a config_directory override requires either root privileges, or it requires that the directory is listed with the alternate_config_directories parameter in the default main.cf file.

connection_cache_protocol_timeout (default: 5s)
Time limit for connection cache connect, send or receive operations. The time limit is enforced in the client.

This feature is available in Postfix 2.3 and later.

connection_cache_service (default: scache)
The name of the **scache**(8) connection cache service. This service maintains a limited pool of cached sessions.

connection_cache_status_update_time (default: 600s)
How frequently the **scache**(8) server logs usage statistics with connection cache hit and miss rates for logical destinations and for physical endpoints.

connection_cache_ttl_limit (default: 2s)
The maximal time-to-live value that the **scache**(8) connection cache server allows. Requests that specify a larger TTL will be stored with the maximum allowed TTL. The purpose of this additional control is to protect the infrastructure against careless people. The cache TTL is already bounded by $max_idle.

content_filter (default: empty)
The name of a mail delivery transport that filters mail after it is queued.

This parameter uses the same syntax as the right-hand side of a Postfix **transport**(5) table. This setting has a lower precedence than a content filter that is specified with an **access**(5) table or in a **header_checks**(5) or **body_checks**(5) table.

daemon_directory (default: see postconf -d output)
The directory with Postfix support programs and daemon programs. These should not be invoked directly by humans. The directory must be owned by root.

daemon_timeout (default: 18000s)
How much time a Postfix daemon process may take to handle a request before it is terminated by a built-in watchdog timer.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

debug_peer_level (default: 2)
The increment in verbose logging level when a remote client or server matches a pattern in the debug_peer_list parameter.

debug_peer_list (default: empty)
Optional list of remote client or server hostname or network address patterns that cause the verbose logging level to increase by the amount specified in $debug_peer_level.

Specify domain names, network/netmask patterns, "/file/name" patterns or "type:table" lookup tables. The right-hand side result from "type:table" lookups is ignored.

Pattern matching of domain names is controlled by the parent_domain_matches_subdomains parameter.

Examples:

```
debug_peer_list = 127.0.0.1
debug_peer_list = some.domain
```

debugger_command (default: empty)

The external command to execute when a Postfix daemon program is invoked with the -D option.

Use "command .. & sleep 5" so that the debugger can attach before the process marches on. If you use an X-based debugger, be sure to set up your XAUTHORITY environment variable before starting Postfix.

Example:

```
debugger_command =
    PATH=/usr/bin:/usr/X11R6/bin
    xxgdb $daemon_directory/$process_name $process_id & sleep 5
```

default_database_type (default: see postconf -d output)

The default database type for use in **newaliases**(1), **postalias**(1) and **postmap**(1) commands. On many UNIX systems the default type is either **dbm** or **hash**. The default setting is frozen when the Postfix system is built.

Examples:

```
default_database_type = hash
default_database_type = dbm
```

default_delivery_slot_cost (default: 5)

How often the Postfix queue manager’s scheduler is allowed to preempt delivery of one message with another.

Each transport maintains a so-called "available delivery slot counter" for each message. One message can be preempted by another one when the other message can be delivered using no more delivery slots (i.e., invocations of delivery agents) than the current message counter has accumulated (or will eventually accumulate - see about slot loans below). This parameter controls how often is the counter incremented - it happens after each default_delivery_slot_cost recipients have been delivered.

The cost of 0 is used to disable the preempting scheduling completely. The minimum value the scheduling algorithm can use is 2 - use it if you want to maximize the message throughput rate. Although there is no maximum, it doesn’t make much sense to use values above say 50.

The only reason why the value of 2 is not the default is the way this parameter affects the delivery of mailing-list mail. In the worst case, their delivery can take somewhere between (cost+1/cost) and (cost/cost-1) times more than if the preemptive scheduler was disabled. The default value of 5 turns out to provide reasonable message response times while making sure the mailing-list deliveries are not extended by more than 20-25 percent even in the worst case.

Examples:

```
default_delivery_slot_cost = 0
default_delivery_slot_cost = 2
```

default_delivery_slot_discount (default: 50)

The default value for transport-specific _delivery_slot_discount settings.

This parameter speeds up the moment when a message preemption can happen. Instead of waiting until the full amount of delivery slots required is available, the preemption can happen when transport_delivery_slot_discount percent of the required amount plus transport_delivery_slot_loan still remains to be accumulated. Note that the full amount will still have to be accumulated before another preemption can take place later.

default_delivery_slot_loan (default: 3)

The default value for transport-specific _delivery_slot_loan settings.

This parameter speeds up the moment when a message preemption can happen. Instead of waiting until the

full amount of delivery slots required is available, the preemption can happen when transport_delivery_slot_discount percent of the required amount plus transport_delivery_slot_loan still remains to be accumulated. Note that the full amount will still have to be accumulated before another preemption can take place later.

default_destination_concurrency_limit (default: 20)

The default maximal number of parallel deliveries to the same destination. This is the default limit for delivery via the **lmtp**(8), **pipe**(8), **smtp**(8) and **virtual**(8) delivery agents.

default_destination_recipient_limit (default: 50)

The default maximal number of recipients per message delivery. This is the default limit for delivery via the **lmtp**(8), **pipe**(8), **smtp**(8) and **virtual**(8) delivery agents.

Setting this parameter to a value of 1 changes the meaning of the corresponding per-destination concurrency limit from concurrency per domain into concurrency per recipient.

default_extra_recipient_limit (default: 1000)

The default value for the extra per-transport limit imposed on the number of in-memory recipients. This extra recipient space is reserved for the cases when the Postfix queue manager's scheduler preempts one message with another and suddenly needs some extra recipients slots for the chosen message in order to avoid performance degradation.

default_minimum_delivery_slots (default: 3)

How many recipients a message must have in order to invoke the Postfix queue manager's scheduling algorithm at all. Messages which would never accumulate at least this many delivery slots (subject to slot cost parameter as well) are never preempted.

default_privs (default: nobody)

The default rights used by the **local**(8) delivery agent for delivery to external file or command. These rights are used when delivery is requested from an **aliases**(5) file that is owned by **root**, or when delivery is done on behalf of **root**. **DO NOT SPECIFY A PRIVILEGED USER OR THE POSTFIX OWNER**.

default_process_limit (default: 100)

The default maximal number of Postfix child processes that provide a given service. This limit can be overruled for specific services in the master.cf file.

default_rbl_reply (default: see postconf -d output)

The default SMTP server response template for a request that is rejected by an RBL-based restriction. This template can be overruled by specific entries in the optional rbl_reply_maps lookup table.

This feature is available in Postfix 2.0 and later.

The template is subject to exactly one level of $name substitution:

$client The client hostname and IP address, formatted as name[address].

$client_address
The client IP address.

$client_name
The client hostname or "unknown". See reject_unknown_client_hostname for more details.

$reverse_client_name
The client hostname from address->name lookup, or "unknown". See reject_unknown_reverse_client_hostname for more details.

$helo_name
The hostname given in HELO or EHLO command or empty string.

$rbl_class
The blacklisted entity type: Client host, Helo command, Sender address, or Recipient address.

$rbl_code
The numerical SMTP response code, as specified with the maps_rbl_reject_code configuration parameter. Note: The numerical SMTP response code is required, and must appear at the start of

the reply. With Postfix version 2.3 and later this information may be followed by an RFC 3463 enhanced status code.

$rbl_domain
The RBL domain where $rbl_what is blacklisted.

$rbl_reason
The reason why $rbl_what is blacklisted, or an empty string.

$rbl_what
The entity that is blacklisted (an IP address, a hostname, a domain name, or an email address whose domain was blacklisted).

$recipient
The recipient address or <> in case of the null address.

$recipient_domain
The recipient domain or empty string.

$recipient_name
The recipient address localpart or <> in case of null address.

$sender
The sender address or <> in case of the null address.

$sender_domain
The sender domain or empty string.

$sender_name
The sender address localpart or <> in case of the null address.

${name?text}
Expands to 'text' if $name is not empty.

${name:text}
Expands to 'text' if $name is empty.

Instead of $name you can also specify ${name} or $(name).

Note: when an enhanced status code is specified in an RBL reply template, it is subject to modification. The following transformations are needed when the same RBL reply template is used for client, helo, sender, or recipient access restrictions.

- When rejecting a sender address, the Postfix SMTP server will transform a recipient DSN status (e.g., 4.1.1-4.1.6) into the corresponding sender DSN status, and vice versa.
- When rejecting non-address information (such as the HELO command argument or the client hostname/address), the Postfix SMTP server will transform a sender or recipient DSN status into a generic non-address DSN status (e.g., 4.0.0).

default_recipient_limit (default: 20000)
The default per-transport upper limit on the number of in-memory recipients. These limits take priority over the global qmgr_message_recipient_limit after the message has been assigned to the respective transports. See also default_extra_recipient_limit and qmgr_message_recipient_minimum.

default_recipient_refill_delay (default: 5s)
The default per-transport maximum delay between recipients refills. When not all message recipients fit into the memory at once, keep loading more of them at least once every this many seconds. This is used to make sure the recipients are refilled in timely manner even when $default_recipient_refill_limit is too high for too slow deliveries.

default_recipient_refill_limit (default: 100)
The default per-transport limit on the number of recipients refilled at once. When not all message recipients fit into the memory at once, keep loading more of them in batches of at least this many at a time. See also $default_recipient_refill_delay, which may result in recipient batches lower than this when this limit is

too high for too slow deliveries.

default_transport (default: smtp)

The default mail delivery transport and next-hop destination for destinations that do not match $mydestination, $inet_interfaces, $proxy_interfaces, $virtual_alias_domains, $virtual_mailbox_domains, or $relay_domains. In order of decreasing precedence, the nexthop destination is taken from $default_transport, $sender_dependent_relayhost_maps, $relayhost, or from the recipient domain. This information can be overruled with the **transport**(5) table.

Specify a string of the form *transport:nexthop*, where *transport* is the name of a mail delivery transport defined in master.cf. The *:nexthop* part is optional. For more details see the **transport**(5) manual page.

Example:

```
default_transport = uucp:relayhostname
```

default_verp_delimiters (default: +=)

The two default VERP delimiter characters. These are used when no explicit delimiters are specified with the SMTP XVERP command or with the "**sendmail -V**" command-line option. Specify characters that are allowed by the verp_delimiter_filter setting.

This feature is available in Postfix 1.1 and later.

defer_code (default: 450)

The numerical Postfix SMTP server response code when a remote SMTP client request is rejected by the "defer" restriction.

Do not change this unless you have a complete understanding of RFC 821.

defer_service_name (default: defer)

The name of the defer service. This service is implemented by the **bounce**(8) daemon and maintains a record of failed delivery attempts and generates non-delivery notifications.

This feature is available in Postfix 2.0 and later.

defer_transports (default: empty)

The names of message delivery transports that should not deliver mail unless someone issues "**sendmail -q**" or equivalent. Specify zero or more names of mail delivery transports names that appear in the first field of master.cf.

Example:

```
defer_transports = smtp
```

delay_logging_resolution_limit (default: 2)

The maximal number of digits after the decimal point when logging sub-second delay values. Specify a number in the range 0..6.

Large delay values are rounded off to an integral number seconds; delay values below the delay_logging_resolution_limit are logged as "0", and small delay values are logged with at most two-digit precision.

The format of the "delays=a/b/c/d" logging is as follows:

- a = time from message arrival to last active queue entry
- b = time from last active queue entry to connection setup
- c = time in connection setup, including DNS, EHLO and TLS
- d = time in message transmission

This feature is available in Postfix 2.3 and later.

delay_notice_recipient (default: postmaster)

The recipient of postmaster notifications with the message headers of mail that cannot be delivered within $delay_warning_time time units.

This feature is enabled with the delay_warning_time parameter.

delay_warning_time (default: 0h)
The time after which the sender receives the message headers of mail that is still queued.

To enable this feature, specify a non-zero time value (an integral value plus an optional one-letter suffix that specifies the time unit).

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is h (hours).

deliver_lock_attempts (default: 20)
The maximal number of attempts to acquire an exclusive lock on a mailbox file or **bounce**(8) logfile.

deliver_lock_delay (default: 1s)
The time between attempts to acquire an exclusive lock on a mailbox file or **bounce**(8) logfile.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

disable_dns_lookups (default: no)
Disable DNS lookups in the Postfix SMTP and LMTP clients. When disabled, hosts are looked up with the getaddrinfo() system library routine which normally also looks in /etc/hosts.

DNS lookups are enabled by default.

disable_mime_input_processing (default: no)
Turn off MIME processing while receiving mail. This means that no special treatment is given to Content-Type: message headers, and that all text after the initial message headers is considered to be part of the message body.

This feature is available in Postfix 2.0 and later.

Mime input processing is enabled by default, and is needed in order to recognize MIME headers in message content.

disable_mime_output_conversion (default: no)
Disable the conversion of 8BITMIME format to 7BIT format. Mime output conversion is needed when the destination does not advertise 8BITMIME support.

This feature is available in Postfix 2.0 and later.

disable_verp_bounces (default: no)
Disable sending one bounce report per recipient.

The default, one per recipient, is what ezmlm needs.

This feature is available in Postfix 1.1 and later.

disable_vrfy_command (default: no)
Disable the SMTP VRFY command. This stops some techniques used to harvest email addresses.

Example:

```
disable_vrfy_command = no
```

dont_remove (default: 0)
Don't remove queue files and save them to the "saved" mail queue. This is a debugging aid. To inspect the envelope information and content of a Postfix queue file, use the **postcat**(1) command.

double_bounce_sender (default: double-bounce)
The sender address of postmaster notifications that are generated by the mail system. All mail to this address is silently discarded, in order to terminate mail bounce loops.

duplicate_filter_limit (default: 1000)
The maximal number of addresses remembered by the address duplicate filter for **aliases**(5) or **virtual**(5) alias expansion, or for **showq**(8) queue displays.

empty_address_recipient (default: MAILER-DAEMON)
The recipient of mail addressed to the null address. Postfix does not accept such addresses in SMTP commands, but they may still be created locally as the result of configuration or software error.

enable_errors_to (default: no)

Report mail delivery errors to the address specified with the non-standard Errors-To: message header, instead of the envelope sender address (this feature is removed with Postfix version 2.2, is turned off by default with Postfix version 2.1, and is always turned on with older Postfix versions).

enable_original_recipient (default: yes)

Enable support for the X-Original-To message header. This header is needed for multi-recipient mailboxes.

When this parameter is set to yes, the **cleanup**(8) daemon performs duplicate elimination on distinct pairs of (original recipient, rewritten recipient), and generates non-empty original recipient queue file records.

When this parameter is set to no, the **cleanup**(8) daemon performs duplicate elimination on the rewritten recipient address only, and generates empty original recipient queue file records.

This feature is available in Postfix 2.1 and later. With Postfix version 2.0, support for the X-Original-To message header is always turned on. Postfix versions before 2.0 have no support for the X-Original-To message header.

error_notice_recipient (default: postmaster)

The recipient of postmaster notifications about mail delivery problems that are caused by policy, resource, software or protocol errors. These notifications are enabled with the notify_classes parameter.

error_service_name (default: error)

The name of the **error**(8) pseudo delivery agent. This service always returns mail as undeliverable.

This feature is available in Postfix 2.0 and later.

execution_directory_expansion_filter (default: see postconf -d output)

Restrict the characters that the **local**(8) delivery agent allows in $name expansions of $command_execution_directory. Characters outside the allowed set are replaced by underscores.

This feature is available in Postfix 2.2 and later.

expand_owner_alias (default: no)

When delivering to an alias "aliasname" that has an "owner-aliasname" companion alias, set the envelope sender address to the expansion of the "owner-aliasname" alias. Normally, Postfix sets the envelope sender address to the name of the "owner-aliasname" alias.

export_environment (default: see postconf -d output)

The list of environment variables that a Postfix process will export to non-Postfix processes. The TZ variable is needed for sane time keeping on System-V-ish systems.

Specify a list of names and/or name=value pairs, separated by whitespace or comma. The name=value form is supported with Postfix version 2.1 and later.

Example:

```
export_environment = TZ PATH=/bin:/usr/bin
```

extract_recipient_limit (default: 10240)

The maximal number of recipient addresses that Postfix will extract from message headers when mail is submitted with "**sendmail -t**".

This feature was removed in Postfix version 2.1.

fallback_relay (default: empty)

Optional list of relay hosts for SMTP destinations that can’t be found or that are unreachable. With Postfix 2.3 this parameter is renamed to smtp_fallback_relay.

By default, mail is returned to the sender when a destination is not found, and delivery is deferred when a destination is unreachable.

The fallback relays must be SMTP destinations. Specify a domain, host, host:port, [host]:port, [address] or [address]:port; the form [host] turns off MX lookups. If you specify multiple SMTP destinations, Postfix will try them in the specified order.

Note: before Postfix 2.2, do not use the fallback_relay feature when relaying mail for a backup or primary MX domain. Mail would loop between the Postfix MX host and the fallback_relay host when the final destination is unavailable.

- In main.cf specify "relay_transport = relay",
- In master.cf specify "-o fallback_relay =" (i.e., empty) at the end of the relay entry.
- In transport maps, specify "relay:*nexthop*..." as the right-hand side for backup or primary MX domain entries.

Postfix version 2.2 and later will not use the fallback_relay feature for destinations that it is MX host for.

fallback_transport (default: empty)

Optional message delivery transport that the **local**(8) delivery agent should use for names that are not found in the **aliases**(5) or UNIX password database.

The precedence of **local**(8) delivery features from high to low is: aliases, .forward files, mailbox_transport_maps, mailbox_transport, mailbox_command_maps, mailbox_command, home_mailbox, mail_spool_directory, fallback_transport_maps, fallback_transport and luser_relay.

fallback_transport_maps (default: empty)

Optional lookup tables with per-recipient message delivery transports for recipients that the **local**(8) delivery agent could not find in the **aliases**(5) or UNIX password database.

The precedence of **local**(8) delivery features from high to low is: aliases, .forward files, mailbox_transport_maps, mailbox_transport, mailbox_command_maps, mailbox_command, home_mailbox, mail_spool_directory, fallback_transport_maps, fallback_transport and luser_relay.

For safety reasons, this feature does not allow $number substitutions in regular expression maps.

This feature is available in Postfix 2.3 and later.

fast_flush_domains (default: $relay_domains)

Optional list of destinations that are eligible for per-destination logfiles with mail that is queued to those destinations.

By default, Postfix maintains "fast flush" logfiles only for destinations that the Postfix SMTP server is willing to relay to (i.e. the default is: "fast_flush_domains = $relay_domains"; see the relay_domains parameter in the **postconf**(5) manual).

Specify a list of hosts or domains, "/file/name" patterns or "type:table" lookup tables, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when the domain or its parent domain appears as lookup key.

Specify "fast_flush_domains =" (i.e., empty) to disable the feature altogether.

fast_flush_purge_time (default: 7d)

The time after which an empty per-destination "fast flush" logfile is deleted.

You can specify the time as a number, or as a number followed by a letter that indicates the time unit: s=seconds, m=minutes, h=hours, d=days, w=weeks. The default time unit is days.

fast_flush_refresh_time (default: 12h)

The time after which a non-empty but unread per-destination "fast flush" logfile needs to be refreshed. The contents of a logfile are refreshed by requesting delivery of all messages listed in the logfile.

You can specify the time as a number, or as a number followed by a letter that indicates the time unit: s=seconds, m=minutes, h=hours, d=days, w=weeks. The default time unit is hours.

fault_injection_code (default: 0)

Force specific internal tests to fail, to test the handling of errors that are difficult to reproduce otherwise.

flush_service_name (default: flush)
The name of the **flush**(8) service. This service maintains per-destination logfiles with the queue file names of mail that is queued for those destinations.

This feature is available in Postfix 2.0 and later.

fork_attempts (default: 5)
The maximal number of attempts to fork() a child process.

fork_delay (default: 1s)
The delay between attempts to fork() a child process.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

forward_expansion_filter (default: see postconf -d output)
Restrict the characters that the **local**(8) delivery agent allows in $name expansions of $forward_path. Characters outside the allowed set are replaced by underscores.

forward_path (default: see postconf -d output)
The **local**(8) delivery agent search list for finding a .forward file with user-specified delivery methods. The first file that is found is used.

The following $name expansions are done on forward_path before the search actually happens. The result of $name expansion is filtered with the character set that is specified with the forward_expansion_filter parameter.

$user The recipient's username.

$shell The recipient's login shell pathname.

$home The recipient's home directory.

$recipient
The full recipient address.

$extension
The optional recipient address extension.

$domain
The recipient domain.

$local The entire recipient localpart.

$recipient_delimiter
The system-wide recipient address extension delimiter.

${name?value}
Expands to *value* when *$name* is non-empty.

${name:value}
Expands to *value* when *$name* is empty.

Instead of $name you can also specify ${name} or $(name).

Examples:

```
forward_path = /var/forward/$user
forward_path =
    /var/forward/$user/.forward$recipient_delimiter$extension,
    /var/forward/$user/.forward
```

frozen_delivered_to (default: yes)
Update the **local**(8) delivery agent's idea of the Delivered-To: address (see prepend_delivered_header) only once, at the start of a delivery attempt; do not update the Delivered-To: address while expanding aliases or .forward files.

This feature is available in Postfix 2.3 and later. With older Postfix releases, the behavior is as if this parameter is set to "no". The old setting can be expensive with deeply nested aliases or .forward files. When an

alias or .forward file changes the Delivered-To: address, it ties up one queue file and one cleanup process instance while mail is being forwarded.

hash_queue_depth (default: 1)
The number of subdirectory levels for queue directories listed with the hash_queue_names parameter.

After changing the hash_queue_names or hash_queue_depth parameter, execute the command "**postfix reload**".

hash_queue_names (default: deferred, defer)
The names of queue directories that are split across multiple subdirectory levels.

Before Postfix version 2.2, the default list of hashed queues was significantly larger. Claims about improvements in file system technology suggest that hashing of the incoming and active queues is no longer needed. Fewer hashed directories speed up the time needed to restart Postfix.

After changing the hash_queue_names or hash_queue_depth parameter, execute the command "**postfix reload**".

header_address_token_limit (default: 10240)
The maximal number of address tokens are allowed in an address message header. Information that exceeds the limit is discarded. The limit is enforced by the **cleanup**(8) server.

header_checks (default: empty)
Optional lookup tables for content inspection of primary non-MIME message headers, as specified in the **header_checks**(5) manual page.

header_size_limit (default: 102400)
The maximal amount of memory in bytes for storing a message header. If a header is larger, the excess is discarded. The limit is enforced by the **cleanup**(8) server.

helpful_warnings (default: yes)
Log warnings about problematic configuration settings, and provide helpful suggestions.

This feature is available in Postfix 2.0 and later.

home_mailbox (default: empty)
Optional pathname of a mailbox file relative to a **local**(8) user's home directory.

Specify a pathname ending in "/" for qmail-style delivery.

The precedence of **local**(8) delivery features from high to low is: aliases, .forward files, mailbox_transport_maps, mailbox_transport, mailbox_command_maps, mailbox_command, home_mailbox, mail_spool_directory, fallback_transport_maps, fallback_transport and luser_relay.

Examples:

```
home_mailbox = Mailbox
home_mailbox = Maildir/
```

hopcount_limit (default: 50)
The maximal number of Received: message headers that is allowed in the primary message headers. A message that exceeds the limit is bounced, in order to stop a mailer loop.

html_directory (default: see postconf -d output)
The location of Postfix HTML files that describe how to build, configure or operate a specific Postfix subsystem or feature.

ignore_mx_lookup_error (default: no)
Ignore DNS MX lookups that produce no response. By default, the Postfix SMTP client defers delivery and tries again after some delay. This behavior is required by the SMTP standard.

Specify "ignore_mx_lookup_error = yes" to force a DNS A record lookup instead. This violates the SMTP standard and can result in mis-delivery of mail.

import_environment (default: see postconf -d output)

The list of environment parameters that a Postfix process will import from a non-Postfix parent process. Examples of relevant parameters:

TZ Needed for sane time keeping on most System-V-ish systems.

DISPLAY
Needed for debugging Postfix daemons with an X-windows debugger.

XAUTHORITY
Needed for debugging Postfix daemons with an X-windows debugger.

MAIL_CONFIG
Needed to make "**postfix -c**" work.

Specify a list of names and/or name=value pairs, separated by whitespace or comma. The name=value form is supported with Postfix version 2.1 and later.

in_flow_delay (default: 1s)

Time to pause before accepting a new message, when the message arrival rate exceeds the message delivery rate. This feature is turned on by default (it's disabled on SCO UNIX due to an SCO bug).

With the default 100 SMTP server process limit, "in_flow_delay = 1s" limits the mail inflow to 100 messages per second above the number of messages delivered per second.

Specify 0 to disable the feature. Valid delays are 0..10.

inet_interfaces (default: all)

The network interface addresses that this mail system receives mail on. Specify "all" to receive mail on all network interfaces (default), and "loopback-only" to receive mail on loopback network interfaces only (Postfix version 2.2 and later). The parameter also controls delivery of mail to user@[ip.address].

Note 1: you need to stop and start Postfix when this parameter changes.

Note 2: address information may be enclosed inside [], but this form is not required here.

When inet_interfaces specifies just one IPv4 and/or IPv6 address that is not a loopback address, the Postfix SMTP client will use this address as the IP source address for outbound mail. Support for IPv6 is available in Postfix version 2.2 and later.

On a multi-homed firewall with separate Postfix instances listening on the "inside" and "outside" interfaces, this can prevent each instance from being able to reach servers on the "other side" of the firewall. Setting smtp_bind_address to 0.0.0.0 avoids the potential problem for IPv4, and setting smtp_bind_address6 to :: solves the problem for IPv6.

A better solution for multi-homed firewalls is to leave inet_interfaces at the default value and instead use explicit IP addresses in the master.cf SMTP server definitions. This preserves the Postfix SMTP client's loop detection, by ensuring that each side of the firewall knows that the other IP address is still the same host. Setting $inet_interfaces to a single IPv4 and/or IPV6 address is primarily useful with virtual hosting of domains on secondary IP addresses, when each IP address serves a different domain (and has a different $myhostname setting).

See also the proxy_interfaces parameter, for network addresses that are forwarded to Postfix by way of a proxy or address translator.

Examples:

```
inet_interfaces = all (DEFAULT)
inet_interfaces = loopback-only (Postfix version 2.2 and later)
inet_interfaces = 127.0.0.1
inet_interfaces = 127.0.0.1, [::1] (Postfix version 2.2 and later)
inet_interfaces = 192.168.1.2, 127.0.0.1
```

inet_protocols (default: ipv4)

The Internet protocols Postfix will attempt to use when making or accepting connections. Specify one or more of "ipv4" or "ipv6", separated by whitespace or commas. The form "all" is equivalent to "ipv4, ipv6" or "ipv4", depending on whether the operating system implements IPv6.

This feature is available in Postfix 2.2 and later.

Note: you MUST stop and start Postfix after changing this parameter.

On systems that pre-date IPV6_V6ONLY support (RFC 3493), an IPv6 server will also accept IPv4 connections, even when IPv4 is turned off with the inet_protocols parameter. On systems with IPV6_V6ONLY support, Postfix will use separate server sockets for IPv6 and IPv4, and each will accept only connections for the corresponding protocol.

When IPv4 support is enabled via the inet_protocols parameter, Postfix will to DNS type A record lookups, and will convert IPv4-in-IPv6 client IP addresses (::ffff:1.2.3.4) to their original IPv4 form (1.2.3.4). The latter is needed on hosts that pre-date IPV6_V6ONLY support (RFC 3493).

When IPv6 support is enabled via the inet_protocols parameter, Postfix will do DNS type AAAA record lookups.

When both IPv4 and IPv6 support are enabled, the Postfix SMTP client will attempt to connect via IPv6 before attempting to use IPv4.

Examples:

```
inet_protocols = ipv4 (DEFAULT)
inet_protocols = all
inet_protocols = ipv6
inet_protocols = ipv4, ipv6
```

initial_destination_concurrency (default: 5)

The initial per-destination concurrency level for parallel delivery to the same destination. This limit applies to delivery via **smtp**(8), and via the **pipe**(8) and **virtual**(8) delivery agents.

Warning: with concurrency of 1, one bad message can be enough to block all mail to a site.

internal_mail_filter_classes (default: empty)

What categories of Postfix-generated mail are subject to before-queue content inspection by non_smtpd_milters, header_checks and body_checks. Specify zero or more of the following, separated by whitespace or comma.

bounce Inspect the content of delivery status notifications.

notify Inspect the content of postmaster notifications by the **smtp**(8) and **smtpd**(8) processes.

NOTE: It's generally not safe to enable content inspection of Postfix-generated email messages. The user is warned.

This feature is available in Postfix 2.3 and later.

invalid_hostname_reject_code (default: 501)

The numerical Postfix SMTP server response code when the client HELO or EHLO command parameter is rejected by the reject_invalid_helo_hostname restriction.

Do not change this unless you have a complete understanding of RFC 821.

ipc_idle (default: version dependent)

The time after which a client closes an idle internal communication channel. The purpose is to allow servers to terminate voluntarily after they become idle. This is used, for example, by the address resolving and rewriting clients.

With Postfix 2.4 the default value was reduced from 100s to 5s.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

ipc_timeout (default: 3600s)
The time limit for sending or receiving information over an internal communication channel. The purpose is to break out of deadlock situations. If the time limit is exceeded the software aborts with a fatal error.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

ipc_ttl (default: 1000s)
The time after which a client closes an active internal communication channel. The purpose is to allow servers to terminate voluntarily after reaching their client limit. This is used, for example, by the address resolving and rewriting clients.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.1 and later.

line_length_limit (default: 2048)
Upon input, long lines are chopped up into pieces of at most this length; upon delivery, long lines are reconstructed.

lmtp_bind_address (default: empty)
The LMTP-specific version of the smtp_bind_address configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_bind_address6 (default: empty)
The LMTP-specific version of the smtp_bind_address6 configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_cache_connection (default: yes)
Keep Postfix LMTP client connections open for up to $max_idle seconds. When the LMTP client receives a request for the same connection the connection is reused.

The effectiveness of cached connections will be determined by the number of LMTP servers in use, and the concurrency limit specified for the LMTP client. Cached connections are closed under any of the following conditions:

- The LMTP client idle time limit is reached. This limit is specified with the Postfix max_idle configuration parameter.
- A delivery request specifies a different destination than the one currently cached.
- The per-process limit on the number of delivery requests is reached. This limit is specified with the Postfix max_use configuration parameter.
- Upon the onset of another delivery request, the LMTP server associated with the current session does not respond to the RSET command.

Most of these limitations will be removed after Postfix implements a connection cache that is shared among multiple LMTP client programs.

lmtp_cname_overrides_servername (default: yes)
The LMTP-specific version of the smtp_cname_overrides_servername configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_connect_timeout (default: 0s)
The LMTP client time limit for completing a TCP connection, or zero (use the operating system built-in time limit). When no connection can be made within the deadline, the LMTP client tries the next address on the mail exchanger list.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

Example:

```
lmtp_connect_timeout = 30s
```

lmtp_connection_cache_destinations (default: empty)
The LMTP-specific version of the smtp_connection_cache_destinations configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_connection_cache_on_demand (default: yes)
The LMTP-specific version of the smtp_connection_cache_on_demand configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_connection_cache_time_limit (default: 2s)
The LMTP-specific version of the smtp_connection_cache_time_limit configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_connection_reuse_time_limit (default: 300s)
The LMTP-specific version of the smtp_connection_reuse_time_limit configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_data_done_timeout (default: 600s)
The LMTP client time limit for sending the LMTP ".", and for receiving the server response. When no response is received within the deadline, a warning is logged that the mail may be delivered multiple times.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_data_init_timeout (default: 120s)
The LMTP client time limit for sending the LMTP DATA command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_data_xfer_timeout (default: 180s)
The LMTP client time limit for sending the LMTP message content. When the connection stalls for more than $lmtp_data_xfer_timeout the LMTP client terminates the transfer.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_defer_if_no_mx_address_found (default: no)
The LMTP-specific version of the smtp_defer_if_no_mx_address_found configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_destination_concurrency_limit (default: $default_destination_concurrency_limit)
The maximal number of parallel deliveries to the same destination via the lmtp message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

lmtp_destination_recipient_limit (default: $default_destination_recipient_limit)
The maximal number of recipients per delivery via the lmtp message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

Setting this parameter to a value of 1 changes the meaning of lmtp_destination_concurrency_limit from concurrency per domain into concurrency per recipient.

lmtp_discard_lhlo_keyword_address_maps (default: empty)
Lookup tables, indexed by the remote LMTP server address, with case insensitive lists of LHLO keywords (pipelining, starttls, auth, etc.) that the LMTP client will ignore in the LHLO response from a remote LMTP server. See lmtp_discard_lhlo_keywords for details. The table is not indexed by hostname for consistency with smtpd_discard_ehlo_keyword_address_maps.

This feature is available in Postfix 2.3 and later.

lmtp_discard_lhlo_keywords (default: empty)
A case insensitive list of LHLO keywords (pipelining, starttls, auth, etc.) that the LMTP client will ignore in the LHLO response from a remote LMTP server.

This feature is available in Postfix 2.3 and later.

Notes:

- Specify the **silent-discard** pseudo keyword to prevent this action from being logged.
- Use the lmtp_discard_lhlo_keyword_address_maps feature to discard LHLO keywords selectively.

lmtp_enforce_tls (default: no)
The LMTP-specific version of the smtp_enforce_tls configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_generic_maps (default: empty)
The LMTP-specific version of the smtp_generic_maps configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_host_lookup (default: dns)
The LMTP-specific version of the smtp_host_lookup configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_lhlo_name (default: $myhostname)
The hostname to send in the LMTP LHLO command.

The default value is the machine hostname. Specify a hostname or [ip.add.re.ss].

This information can be specified in the main.cf file for all LMTP clients, or it can be specified in the master.cf file for a specific client, for example:

```
/etc/postfix/master.cf:
      mylmtp ... lmtp -o lmtp_lhlo_name=foo.bar.com
```

This feature is available in Postfix 2.3 and later.

lmtp_lhlo_timeout (default: 300s)
The LMTP client time limit for sending the LHLO command, and for receiving the initial server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_line_length_limit (default: 990)
The LMTP-specific version of the smtp_line_length_limit configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_mail_timeout (default: 300s)
The LMTP client time limit for sending the MAIL FROM command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_mx_address_limit (default: 5)
The LMTP-specific version of the smtp_mx_address_limit configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_mx_session_limit (default: 2)
The LMTP-specific version of the smtp_mx_session_limit configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_pix_workaround_delay_time (default: 10s)
The LMTP-specific version of the smtp_pix_workaround_delay_time configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_pix_workaround_maps (default: empty)
The LMTP-specific version of the smtp_pix_workaround_maps configuration parameter. See there for details.

This feature is available in Postfix 2.4 and later.

lmtp_pix_workaround_threshold_time (default: 500s)
The LMTP-specific version of the smtp_pix_workaround_threshold_time configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_pix_workarounds (default: empty)
The LMTP-specific version of the smtp_pix_workaround configuration parameter. See there for details.

This feature is available in Postfix 2.4 and later.

lmtp_quit_timeout (default: 300s)
The LMTP client time limit for sending the QUIT command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_quote_rfc821_envelope (default: yes)
The LMTP-specific version of the smtp_quote_rfc821_envelope configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_randomize_addresses (default: yes)
The LMTP-specific version of the smtp_randomize_addresses configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_rcpt_timeout (default: 300s)
The LMTP client time limit for sending the RCPT TO command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_rset_timeout (default: 20s)
The LMTP client time limit for sending the RSET command, and for receiving the server response. The LMTP client sends RSET in order to finish a recipient address probe, or to verify that a cached connection is still alive.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

lmtp_sasl_auth_enable (default: no)
Enable SASL authentication in the Postfix LMTP client.

lmtp_sasl_mechanism_filter (default: empty)
The LMTP-specific version of the smtp_sasl_mechanism_filter configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_sasl_password_maps (default: empty)
Optional LMTP client lookup tables with one username:password entry per host or domain. If a remote host or domain has no username:password entry, then the Postfix LMTP client will not attempt to authenticate to the remote host.

lmtp_sasl_path (default: empty)
Implementation-specific information that is passed through to the SASL plug-in implementation that is selected with **lmtp_sasl_type**. Typically this specifies the name of a configuration file or rendezvous point.

This feature is available in Postfix 2.3 and later.

lmtp_sasl_security_options (default: noplaintext, noanonymous)
SASL security options; as of Postfix 2.3 the list of available features depends on the SASL client implementation that is selected with **lmtp_sasl_type**.

The following security features are defined for the **cyrus** client SASL implementation:

noplaintext
Disallow authentication methods that use plaintext passwords.

noactive
Disallow authentication methods that are vulnerable to non-dictionary active attacks.

nodictionary
Disallow authentication methods that are vulnerable to passive dictionary attack.

noanonymous
Disallow anonymous logins.

Example:

```
lmtp_sasl_security_options = noplaintext
```

lmtp_sasl_tls_security_options (default: $lmtp_sasl_security_options)
The LMTP-specific version of the smtp_sasl_tls_security_options configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_sasl_tls_verified_security_options (default: $lmtp_sasl_tls_security_options)
The LMTP-specific version of the smtp_sasl_tls_verified_security_options configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_sasl_type (default: cyrus)
The SASL plug-in type that the Postfix LMTP client should use for authentication. The available types are listed with the "**postconf -A**" command.

This feature is available in Postfix 2.3 and later.

lmtp_send_xforward_command (default: no)
Send an XFORWARD command to the LMTP server when the LMTP LHLO server response announces XFORWARD support. This allows an **lmtp**(8) delivery agent, used for content filter message injection, to forward the name, address, protocol and HELO name of the original client to the content filter and downstream queuing LMTP server. Before you change the value to yes, it is best to make sure that your content filter supports this command.

This feature is available in Postfix 2.1 and later.

lmtp_sender_dependent_authentication (default: no)
The LMTP-specific version of the smtp_sender_dependent_authentication configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_skip_5xx_greeting (default: yes)
The LMTP-specific version of the smtp_skip_5xx_greeting configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_skip_quit_response (default: no)
Wait for the response to the LMTP QUIT command.

lmtp_starttls_timeout (default: 300s)
The LMTP-specific version of the smtp_starttls_timeout configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tcp_port (default: 24)
The default TCP port that the Postfix LMTP client connects to.

lmtp_tls_CAfile (default: empty)
The LMTP-specific version of the smtp_tls_CAfile configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_CApath (default: empty)
The LMTP-specific version of the smtp_tls_CApath configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_cert_file (default: empty)
The LMTP-specific version of the smtp_tls_cert_file configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_dcert_file (default: empty)
The LMTP-specific version of the smtp_tls_dcert_file configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_dkey_file (default: $lmtp_tls_dcert_file)
The LMTP-specific version of the smtp_tls_dkey_file configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_enforce_peername (default: yes)
The LMTP-specific version of the smtp_tls_enforce_peername configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_exclude_ciphers (default: empty)
The LMTP-specific version of the smtp_tls_exclude_ciphers configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_key_file (default: $lmtp_tls_cert_file)
The LMTP-specific version of the smtp_tls_key_file configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_loglevel (default: 0)
The LMTP-specific version of the smtp_tls_loglevel configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_mandatory_ciphers (default: empty)
The LMTP-specific version of the smtp_tls_mandatory_ciphers configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_mandatory_exclude_ciphers (default: empty)
The LMTP-specific version of the smtp_tls_mandatory_exclude_ciphers configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_mandatory_protocols (default: SSLv3, TLSv1)
The LMTP-specific version of the smtp_tls_mandatory_protocols configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_note_starttls_offer (default: no)
The LMTP-specific version of the smtp_tls_note_starttls_offer configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_per_site (default: empty)
The LMTP-specific version of the smtp_tls_per_site configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_policy_maps (default: empty)
The LMTP-specific version of the smtp_tls_policy_maps configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_scert_verifydepth (default: 5)
The LMTP-specific version of the smtp_tls_scert_verifydepth configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_secure_cert_match (default: nexthop)
The LMTP-specific version of the smtp_tls_secure_cert_match configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_session_cache_database (default: empty)
The LMTP-specific version of the smtp_tls_session_cache_database configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_session_cache_timeout (default: 3600s)
The LMTP-specific version of the smtp_tls_session_cache_timeout configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_tls_verify_cert_match (default: hostname)
The LMTP-specific version of the smtp_tls_verify_cert_match configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_use_tls (default: no)
The LMTP-specific version of the smtp_use_tls configuration parameter. See there for details.

This feature is available in Postfix 2.3 and later.

lmtp_xforward_timeout (default: 300s)
The LMTP client time limit for sending the XFORWARD command, and for receiving the server response.

In case of problems the client does NOT try the next address on the mail exchanger list.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.1 and later.

local_command_shell (default: empty)
Optional shell program for **local**(8) delivery to non-Postfix command. By default, non-Postfix commands are executed directly; commands are given to given to /bin/sh only when they contain shell meta characters or shell built-in commands.

"sendmail’s restricted shell" (smrsh) is what most people will use in order to restrict what programs can be run from e.g. .forward files (smrsh is part of the Sendmail distribution).

Note: when a shell program is specified, it is invoked even when the command contains no shell built-in

commands or meta characters.

Example:

```
local_command_shell = /some/where/smrsh -c
```

local_destination_concurrency_limit (default: 2)

The maximal number of parallel deliveries via the local mail delivery transport to the same recipient (when "local_destination_recipient_limit = 1") or the maximal number of parallel deliveries to the same local domain (when "local_destination_recipient_limit > 1"). This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

A low limit of 2 is recommended, just in case someone has an expensive shell command in a .forward file or in an alias (e.g., a mailing list manager). You don't want to run lots of those at the same time.

local_destination_recipient_limit (default: 1)

The maximal number of recipients per message delivery via the local mail delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

Setting this parameter to a value > 1 changes the meaning of local_destination_concurrency_limit from concurrency per recipient into concurrency per domain.

local_header_rewrite_clients (default: permit_inet_interfaces)

Rewrite message header addresses in mail from these clients and update incomplete addresses with the domain name in $myorigin or $mydomain; either don't rewrite message headers from other clients at all, or rewrite message headers and update incomplete addresses with the domain specified in the remote_header_rewrite_domain parameter.

See the append_at_myorigin and append_dot_mydomain parameters for details of how domain names are appended to incomplete addresses.

Specify a list of zero or more of the following:

permit_inet_interfaces

Append the domain name in $myorigin or $mydomain when the client IP address matches $inet_interfaces. This is enabled by default.

permit_mynetworks

Append the domain name in $myorigin or $mydomain when the client IP address matches any network or network address listed in $mynetworks. This setting will not prevent remote mail header address rewriting when mail from a remote client is forwarded by a neighboring system.

permit_sasl_authenticated

Append the domain name in $myorigin or $mydomain when the client is successfully authenticated via the RFC 2554 (AUTH) protocol.

permit_tls_clientcerts

Append the domain name in $myorigin or $mydomain when the client TLS certificate is successfully verified, and the client certificate fingerprint is listed in $relay_clientcerts.

permit_tls_all_clientcerts

Append the domain name in $myorigin or $mydomain when the client TLS certificate is successfully verified, regardless of whether it is listed on the server, and regardless of the certifying authority.

check_address_map *type:table*

type:table

Append the domain name in $myorigin or $mydomain when the client IP address matches the specified lookup table. The lookup result is ignored, and no subnet lookup is done. This is suitable for, e.g., pop-before-smtp lookup tables.

Examples:

The Postfix < 2.2 backwards compatible setting: always rewrite message headers, and always append my own domain to incomplete header addresses.

```
local_header_rewrite_clients = static:all
```

The purist (and default) setting: rewrite headers only in mail from Postfix sendmail and in SMTP mail from this machine.

```
local_header_rewrite_clients = permit_inet_interfaces
```

The intermediate setting: rewrite header addresses and append $myorigin or $mydomain information only with mail from Postfix sendmail, from local clients, or from authorized SMTP clients.

Note: this setting will not prevent remote mail header address rewriting when mail from a remote client is forwarded by a neighboring system.

```
local_header_rewrite_clients = permit_mynetworks,
    permit_sasl_authenticated permit_tls_clientcerts
    check_address_map hash:/etc/postfix/pop-before-smtp
```

local_recipient_maps (default: proxy:unix:passwd.byname $alias_maps)

Lookup tables with all names or addresses of local recipients: a recipient address is local when its domain matches $mydestination, $inet_interfaces or $proxy_interfaces. Specify @domain as a wild-card for domains that do not have a valid recipient list. Technically, tables listed with $local_recipient_maps are used as lists: Postfix needs to know only if a lookup string is found or not, but it does not use the result from table lookup.

If this parameter is non-empty (the default), then the Postfix SMTP server will reject mail for unknown local users.

To turn off local recipient checking in the Postfix SMTP server, specify "local_recipient_maps =" (i.e. empty).

The default setting assumes that you use the default Postfix local delivery agent for local delivery. You need to update the local_recipient_maps setting if:

- You redefine the local delivery agent in master.cf.
- You redefine the "local_transport" setting in main.cf.
- You use the "luser_relay", "mailbox_transport", or "fallback_transport" feature of the Postfix **local**(8) delivery agent.

Details are described in the LOCAL_RECIPIENT_README file.

Beware: if the Postfix SMTP server runs chrooted, you need to access the passwd file via the **proxymap**(8) service, in order to overcome chroot access restrictions. The alternative, maintaining a copy of the system password file in the chroot jail is not practical.

Examples:

```
local_recipient_maps =
```

local_transport (default: local:$myhostname)

The default mail delivery transport and next-hop destination for final delivery to domains listed with mydestination, and for [ipaddress] destinations that match $inet_interfaces or $proxy_interfaces. This information can be overruled with the **transport**(5) table.

By default, local mail is delivered to the transport called "local", which is just the name of a service that is defined the master.cf file.

Specify a string of the form *transport:nexthop*, where *transport* is the name of a mail delivery transport defined in master.cf. The *:nexthop* part is optional. For more details see the **transport**(5) manual page.

Beware: if you override the default local delivery agent then you need to review the LOCAL_RECIPIENT_README document, otherwise the SMTP server may reject mail for local recipients.

luser_relay (default: empty)

Optional catch-all destination for unknown **local**(8) recipients. By default, mail for unknown recipients in domains that match $mydestination, $inet_interfaces or $proxy_interfaces is returned as undeliverable.

The following $name expansions are done on luser_relay:

$domain
The recipient domain.

$extension
The recipient address extension.

$home The recipient's home directory.

$local The entire recipient address localpart.

$recipient
The full recipient address.

$recipient_delimiter
The system-wide recipient address extension delimiter.

$shell The recipient's login shell.

$user The recipient username.

${name?value}
Expands to *value* when *$name* has a non-empty value.

${name:value}
Expands to *value* when *$name* has an empty value.

Instead of $name you can also specify ${name} or $(name).

Note: luser_relay works only for the Postfix **local**(8) delivery agent.

Note: if you use this feature for accounts not in the UNIX password file, then you must specify "local_recipient_maps =" (i.e. empty) in the main.cf file, otherwise the Postfix SMTP server will reject mail for non-UNIX accounts with "User unknown in local recipient table".

Examples:

```
luser_relay = $user@other.host
luser_relay = $local@other.host
luser_relay = admin+$local
```

mail_name (default: Postfix)

The mail system name that is displayed in Received: headers, in the SMTP greeting banner, and in bounced mail.

mail_owner (default: postfix)

The UNIX system account that owns the Postfix queue and most Postfix daemon processes. Specify the name of a user account that does not share a group with other accounts and that owns no other files or processes on the system. In particular, don't specify nobody or daemon. PLEASE USE A DEDICATED USER ID AND GROUP ID.

When this parameter value is changed you need to re-run "**postfix set-permissions**" (with Postfix version 2.0 and earlier: "**/etc/postfix/post-install set-permissions**".

mail_release_date (default: see postconf -d output)

The Postfix release date, in "YYYYMMDD" format.

mail_spool_directory (default: see postconf -d output)

The directory where **local**(8) UNIX-style mailboxes are kept. The default setting depends on the system type. Specify a name ending in / for maildir-style delivery.

Note: maildir delivery is done with the privileges of the recipient. If you use the mail_spool_directory setting for maildir style delivery, then you must create the top-level maildir directory in advance. Postfix will

not create it.

Examples:

```
mail_spool_directory = /var/mail
mail_spool_directory = /var/spool/mail
```

mail_version (default: see postconf -d output)

The version of the mail system. Stable releases are named *major*.*minor*.*patchlevel*. Experimental releases also include the release date. The version string can be used in, for example, the SMTP greeting banner.

mailbox_command (default: empty)

Optional external command that the **local**(8) delivery agent should use for mailbox delivery. The command is run with the user ID and the primary group ID privileges of the recipient. Exception: command delivery for root executes with $default_privs privileges. This is not a problem, because 1) mail for root should always be aliased to a real user and 2) don't log in as root, use "su" instead.

The following environment variables are exported to the command:

CLIENT_ADDRESS
Remote client network address. Available in Postfix version 2.2 and later.

CLIENT_HELO
Remote client EHLO command parameter. Available in Postfix version 2.2 and later.

CLIENT_HOSTNAME
Remote client hostname. Available in Postfix version 2.2 and later.

CLIENT_PROTOCOL
Remote client protocol. Available in Postfix version 2.2 and later.

DOMAIN
The domain part of the recipient address.

EXTENSION
The optional address extension.

HOME
The recipient home directory.

LOCAL
The recipient address localpart.

LOGNAME
The recipient's username.

RECIPIENT
The full recipient address.

SASL_METHOD
SASL authentication method specified in the remote client AUTH command. Available in Postfix version 2.2 and later.

SASL_SENDER
SASL sender address specified in the remote client MAIL FROM command. Available in Postfix version 2.2 and later.

SASL_USER
SASL username specified in the remote client AUTH command. Available in Postfix version 2.2 and later.

SENDER
The full sender address.

SHELL
The recipient's login shell.

USER The recipient username.

Unlike other Postfix configuration parameters, the mailbox_command parameter is not subjected to $name substitutions. This is to make it easier to specify shell syntax (see example below).

If you can, avoid shell meta characters because they will force Postfix to run an expensive shell process. If you're delivering via Procmail then running a shell won't make a noticeable difference in the total cost.

Note: if you use the mailbox_command feature to deliver mail system-wide, you must set up an alias that forwards mail for root to a real user.

The precedence of **local**(8) delivery features from high to low is: aliases, .forward files, mailbox_transport_maps, mailbox_transport, mailbox_command_maps, mailbox_command, home_mailbox, mail_spool_directory, fallback_transport_maps, fallback_transport and luser_relay.

Examples:

```
mailbox_command = /some/where/procmail
mailbox_command = /some/where/procmail -a "$EXTENSION"
mailbox_command = /some/where/maildrop -d "$USER"
        -f "$SENDER" "$EXTENSION"
```

mailbox_command_maps (default: empty)
Optional lookup tables with per-recipient external commands to use for **local**(8) mailbox delivery. Behavior is as with mailbox_command.

The precedence of **local**(8) delivery features from high to low is: aliases, .forward files, mailbox_transport_maps, mailbox_transport, mailbox_command_maps, mailbox_command, home_mailbox, mail_spool_directory, fallback_transport_maps, fallback_transport and luser_relay.

mailbox_delivery_lock (default: see postconf -d output)
How to lock a UNIX-style **local**(8) mailbox before attempting delivery. For a list of available file locking methods, use the "**postconf -l**" command.

This setting is ignored with **maildir** style delivery, because such deliveries are safe without explicit locks.

Note: The **dotlock** method requires that the recipient UID or GID has write access to the parent directory of the mailbox file.

Note: the default setting of this parameter is system dependent.

mailbox_size_limit (default: 51200000)
The maximal size of any **local**(8) individual mailbox or maildir file, or zero (no limit). In fact, this limits the size of any file that is written to upon local delivery, including files written by external commands that are executed by the **local**(8) delivery agent.

This limit must not be smaller than the message size limit.

mailbox_transport (default: empty)
Optional message delivery transport that the **local**(8) delivery agent should use for mailbox delivery to all local recipients, whether or not they are found in the UNIX passwd database.

The precedence of **local**(8) delivery features from high to low is: aliases, .forward files, mailbox_transport_maps, mailbox_transport, mailbox_command_maps, mailbox_command, home_mailbox, mail_spool_directory, fallback_transport_maps, fallback_transport and luser_relay.

mailbox_transport_maps (default: empty)
Optional lookup tables with per-recipient message delivery transports to use for **local**(8) mailbox delivery, whether or not the recipients are found in the UNIX passwd database.

The precedence of **local**(8) delivery features from high to low is: aliases, .forward files, mailbox_transport_maps, mailbox_transport, mailbox_command_maps, mailbox_command, home_mailbox, mail_spool_directory, fallback_transport_maps, fallback_transport and luser_relay.

For safety reasons, this feature does not allow $number substitutions in regular expression maps.

This feature is available in Postfix 2.3 and later.

mailq_path (default: see postconf -d output)
Sendmail compatibility feature that specifies where the Postfix **mailq**(1) command is installed. This command can be used to list the Postfix mail queue.

manpage_directory (default: see postconf -d output)
Where the Postfix manual pages are installed.

maps_rbl_domains (default: empty)
Obsolete feature: use the reject_rbl_client feature instead.

maps_rbl_reject_code (default: 554)
The numerical Postfix SMTP server response code when a remote SMTP client request is blocked by the reject_rbl_client, reject_rhsbl_client, reject_rhsbl_sender or reject_rhsbl_recipient restriction.

Do not change this unless you have a complete understanding of RFC 821.

masquerade_classes (default: envelope_sender, header_sender, header_recipient)
What addresses are subject to address masquerading.

By default, address masquerading is limited to envelope sender addresses, and to header sender and header recipient addresses. This allows you to use address masquerading on a mail gateway while still being able to forward mail to users on individual machines.

Specify zero or more of: envelope_sender, envelope_recipient, header_sender, header_recipient

masquerade_domains (default: empty)
Optional list of domains whose subdomain structure will be stripped off in email addresses.

The list is processed left to right, and processing stops at the first match. Thus,

```
masquerade_domains = foo.example.com example.com
```

strips "user@any.thing.foo.example.com" to "user@foo.example.com", but strips "user@any.thing.else.example.com" to "user@example.com".

A domain name prefixed with ! means do not masquerade this domain or its subdomains. Thus,

```
masquerade_domains = !foo.example.com example.com
```

does not change "user@any.thing.foo.example.com" or "user@foo.example.com", but strips "user@any.thing.else.example.com" to "user@example.com".

Note: with Postfix version 2.2, message header address masquerading happens only when message header address rewriting is enabled:

- The message is received with the Postfix **sendmail**(1) command,
- The message is received from a network client that matches $local_header_rewrite_clients,
- The message is received from the network, and the remote_header_rewrite_domain parameter specifies a non-empty value.

To get the behavior before Postfix version 2.2, specify "local_header_rewrite_clients = static:all".

Example:

```
masquerade_domains = $mydomain
```

masquerade_exceptions (default: empty)
Optional list of user names that are not subjected to address masquerading, even when their address matches $masquerade_domains.

By default, address masquerading makes no exceptions.

Specify a list of user names, "/file/name" or "type:table" patterns, separated by commas and/or whitespace. The list is matched left to right, and the search stops on the first match. A "/file/name" pattern is replaced by

its contents; a "type:table" lookup table is matched when a name matches a lookup key (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude a name from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Examples:

```
masquerade_exceptions = root, mailer-daemon
masquerade_exceptions = root
```

max_idle (default: 100s)
The maximum amount of time that an idle Postfix daemon process waits for an incoming connection before terminating voluntarily. This parameter is ignored by the Postfix queue manager and by other long-lived Postfix daemon processes.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

max_use (default: 100)
The maximal number of incoming connections that a Postfix daemon process will service before terminating voluntarily. This parameter is ignored by the Postfix queue manager and by other long-lived Postfix daemon processes.

maximal_backoff_time (default: 4000s)
The maximal time between attempts to deliver a deferred message.

This parameter should be set to a value greater than or equal to $minimal_backoff_time. See also $queue_run_delay.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

maximal_queue_lifetime (default: 5d)
The maximal time a message is queued before it is sent back as undeliverable.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is d (days).

Specify 0 when mail delivery should be tried only once.

message_reject_characters (default: empty)
The set of characters that Postfix will reject in message content. The usual C-like escape sequences are recognized: \a \b \f \n \r \t \v *ddd* (up to three octal digits) and \\.

Example:

```
message_reject_characters = \0
```

This feature is available in Postfix 2.3 and later.

message_size_limit (default: 10240000)
The maximal size in bytes of a message, including envelope information.

message_strip_characters (default: empty)
The set of characters that Postfix will remove from message content. The usual C-like escape sequences are recognized: \a \b \f \n \r \t \v *ddd* (up to three octal digits) and \\.

Example:

```
message_strip_characters = \0
```

This feature is available in Postfix 2.3 and later.

milter_command_timeout (default: 30s)
The time limit for sending an SMTP command to a Milter (mail filter) application, and for receiving the response.

Specify a non-zero time value (an integral value plus an optional one-letter suffix that specifies the time unit).

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.3 and later.

milter_connect_macros (default: see postconf -n output)
The macros that are sent to Milter (mail filter) applications after completion of an SMTP connection. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_connect_timeout (default: 30s)
The time limit for connecting to a Milter (mail filter) application, and for negotiating protocol options.

Specify a non-zero time value (an integral value plus an optional one-letter suffix that specifies the time unit).

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.3 and later.

milter_content_timeout (default: 300s)
The time limit for sending message content to a Milter (mail filter) application, and for receiving the response.

Specify a non-zero time value (an integral value plus an optional one-letter suffix that specifies the time unit).

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.3 and later.

milter_data_macros (default: see postconf -n output)
The macros that are sent to version 4 or higher Milter (mail filter) applications after the SMTP DATA command. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_default_action (default: tempfail)
The default action when a Milter (mail filter) application is unavailable or mis-configured. Specify one of the following:

accept Proceed as if the mail filter was not present.

reject Reject all further commands in this session with a permanent status code.

tempfail
Reject all further commands in this session with a temporary status code.

This feature is available in Postfix 2.3 and later.

milter_end_of_data_macros (default: see postconf -n output)
The macros that are sent to Milter (mail filter) applications after the message end-of-data. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_helo_macros (default: see postconf -n output)
The macros that are sent to Milter (mail filter) applications after the SMTP HELO or EHLO command. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_macro_daemon_name (default: $myhostname)
The {daemon_name} macro value for Milter (mail filter) applications. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_macro_v (default: $mail_name $mail_version)
The {v} macro value for Milter (mail filter) applications. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_mail_macros (default: see postconf -n output)
The macros that are sent to Milter (mail filter) applications after the SMTP MAIL FROM command. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_protocol (default: 2)
The mail filter protocol version and optional protocol extensions for communication with a Milter (mail filter) application. This information should match the protocol that is expected by the actual mail filter application.

Protocol versions:

2 Use Sendmail 8 mail filter protocol version 2.

3 Use Sendmail 8 mail filter protocol version 3.

4 Use Sendmail 8 mail filter protocol version 4.

Protocol extensions:

no_header_reply
Specify this when the Milter application will not reply for each individual message header.

This feature is available in Postfix 2.3 and later.

milter_rcpt_macros (default: see postconf -n output)
The macros that are sent to Milter (mail filter) applications after the SMTP RCPT TO command. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

milter_unknown_command_macros (default: see postconf -n output)
The macros that are sent to version 3 or higher Milter (mail filter) applications after an unknown SMTP command. See MILTER_README for a list of available macro names and their meanings.

This feature is available in Postfix 2.3 and later.

mime_boundary_length_limit (default: 2048)
The maximal length of MIME multipart boundary strings. The MIME processor is unable to distinguish between boundary strings that do not differ in the first $mime_boundary_length_limit characters.

This feature is available in Postfix 2.0 and later.

mime_header_checks (default: $header_checks)
Optional lookup tables for content inspection of MIME related message headers, as described in the **header_checks**(5) manual page.

This feature is available in Postfix 2.0 and later.

mime_nesting_limit (default: 100)
The maximal recursion level that the MIME processor will handle. Postfix refuses mail that is nested deeper than the specified limit.

This feature is available in Postfix 2.0 and later.

minimal_backoff_time (default: 300s)
The minimal time between attempts to deliver a deferred message; prior to Postfix 2.4 the default value was 1000s.

This parameter also limits the time an unreachable destination is kept in the short-term, in-memory, destination status cache.

This parameter should be set greater than or equal to $queue_run_delay. See also $maximal_backoff_time.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

multi_recipient_bounce_reject_code (default: 550)

The numerical Postfix SMTP server response code when a remote SMTP client request is blocked by the reject_multi_recipient_bounce restriction.

Do not change this unless you have a complete understanding of RFC 821.

This feature is available in Postfix 2.1 and later.

mydestination (default: $myhostname, localhost.$mydomain, localhost)

The list of domains that are delivered via the $local_transport mail delivery transport. By default this is the Postfix **local**(8) delivery agent which looks up all recipients in /etc/passwd and /etc/aliases. The SMTP server validates recipient addresses with $local_recipient_maps and rejects non-existent recipients. See also the local domain class in the ADDRESS_CLASS_README file.

The default mydestination value specifies names for the local machine only. On a mail domain gateway, you should also include $mydomain.

The $local_transport delivery method is also selected for mail addressed to user@[the.net.work.address] of the mail system (the IP addresses specified with the inet_interfaces and proxy_interfaces parameters).

Warnings:

- Do not specify the names of virtual domains - those domains are specified elsewhere. See VIRTUAL_README for more information.
- Do not specify the names of domains that this machine is backup MX host for. See STANDARD_CONFIGURATION_README for how to set up backup MX hosts.
- By default, the Postfix SMTP server rejects mail for recipients not listed with the local_recipient_maps parameter. See the **postconf**(5) manual for a description of the local_recipient_maps and unknown_local_recipient_reject_code parameters.

Specify a list of host or domain names, "/file/name" or "type:table" patterns, separated by commas and/or whitespace. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a name matches a lookup key (the lookup result is ignored). Continue long lines by starting the next line with whitespace.

Examples:

```
mydestination = $myhostname, localhost.$mydomain $mydomain
mydestination = $myhostname, localhost.$mydomain www.$mydomain, ftp.$mydomain
```

mydomain (default: see postconf -d output)

The internet domain name of this mail system. The default is to use $myhostname minus the first component. $mydomain is used as a default value for many other configuration parameters.

Example:

```
mydomain = domain.tld
```

myhostname (default: see postconf -d output)

The internet hostname of this mail system. The default is to use the fully-qualified domain name from gethostname(). $myhostname is used as a default value for many other configuration parameters.

Example:

```
myhostname = host.domain.tld
```

mynetworks (default: see postconf -d output)

The list of "trusted" SMTP clients that have more privileges than "strangers".

In particular, "trusted" SMTP clients are allowed to relay mail through Postfix. See the smtpd_recipient_restrictions parameter description in the **postconf**(5) manual.

You can specify the list of "trusted" network addresses by hand or you can let Postfix do it for you (which is the default). See the description of the mynetworks_style parameter for more information.

If you specify the mynetworks list by hand, Postfix ignores the mynetworks_style setting.

Specify a list of network addresses or network/netmask patterns, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace.

The netmask specifies the number of bits in the network part of a host address. You can also specify "/file/name" or "type:table" patterns. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a table entry matches a lookup string (the lookup result is ignored).

The list is matched left to right, and the search stops on the first match. Specify "!pattern" to exclude an address or network block from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Note: IP version 6 address information must be specified inside [] in the mynetworks value, and in files specified with "/file/name". IP version 6 addresses contain the ":" character, and would otherwise be confused with a "type:table" pattern.

Examples:

```
mynetworks = 127.0.0.0/8 168.100.189.0/28
mynetworks = !192.168.0.1, 192.168.0.0/28
mynetworks = 127.0.0.0/8 168.100.189.0/28 [::1]/128 [2001:240:587::]/64
mynetworks = $config_directory/mynetworks
mynetworks = hash:/etc/postfix/network_table
```

mynetworks_style (default: subnet)

The method to generate the default value for the mynetworks parameter. This is the list of trusted networks for relay access control etc.

- Specify "mynetworks_style = host" when Postfix should "trust" only the local machine.
- Specify "mynetworks_style = subnet" when Postfix should "trust" SMTP clients in the same IP subnetworks as the local machine. On Linux, this works correctly only with interfaces specified with the "ifconfig" command.
- Specify "mynetworks_style = class" when Postfix should "trust" SMTP clients in the same IP class A/B/C networks as the local machine. Don’t do this with a dialup site - it would cause Postfix to "trust" your entire provider’s network. Instead, specify an explicit mynetworks list by hand, as described with the mynetworks configuration parameter.

myorigin (default: $myhostname)

The domain name that locally-posted mail appears to come from, and that locally posted mail is delivered to. The default, $myhostname, is adequate for small sites. If you run a domain with multiple machines, you should (1) change this to $mydomain and (2) set up a domain-wide alias database that aliases each user to user@that.users.mailhost.

Example:

```
myorigin = $mydomain
```

nested_header_checks (default: $header_checks)

Optional lookup tables for content inspection of non-MIME message headers in attached messages, as described in the **header_checks**(5) manual page.

This feature is available in Postfix 2.0 and later.

newaliases_path (default: see postconf -d output)

Sendmail compatibility feature that specifies the location of the **newaliases**(1) command. This command can be used to rebuild the **local**(8) **aliases**(5) database.

non_fqdn_reject_code (default: 504)

The numerical Postfix SMTP server reply code when a client request is rejected by the reject_non_fqdn_helo_hostname, reject_non_fqdn_sender or reject_non_fqdn_recipient restriction.

non_smtpd_milters (default: empty)

A list of Milter (mail filter) applications for new mail that does not arrive via the Postfix **smtpd**(8) server. This includes local submission via the **sendmail**(1) command line, new mail that arrives via the Postfix

qmqpd(8) server, and old mail that is re-injected into the queue with "postsuper -r". See the MILTER_README document for details.

This feature is available in Postfix 2.3 and later.

notify_classes (default: resource, software)

The list of error classes that are reported to the postmaster. The default is to report only the most serious problems. The paranoid may wish to turn on the policy (UCE and mail relaying) and protocol error (broken mail software) reports.

NOTE: postmaster notifications may contain confidential information such as SASL passwords or message content. It is the system administrator's responsibility to treat such information with care.

The error classes are:

bounce (also implies **2bounce**)
: Send the postmaster copies of the headers of bounced mail, and send transcripts of SMTP sessions when Postfix rejects mail. The notification is sent to the address specified with the bounce_notice_recipient configuration parameter (default: postmaster).

2bounce
: Send undeliverable bounced mail to the postmaster. The notification is sent to the address specified with the 2bounce_notice_recipient configuration parameter (default: postmaster).

delay Send the postmaster copies of the headers of delayed mail. The notification is sent to the address specified with the delay_notice_recipient configuration parameter (default: postmaster).

policy Send the postmaster a transcript of the SMTP session when a client request was rejected because of (UCE) policy. The notification is sent to the address specified with the error_notice_recipient configuration parameter (default: postmaster).

protocol
: Send the postmaster a transcript of the SMTP session in case of client or server protocol errors. The notification is sent to the address specified with the error_notice_recipient configuration parameter (default: postmaster).

resource
: Inform the postmaster of mail not delivered due to resource problems. The notification is sent to the address specified with the error_notice_recipient configuration parameter (default: postmaster).

software
: Inform the postmaster of mail not delivered due to software problems. The notification is sent to the address specified with the error_notice_recipient configuration parameter (default: postmaster).

Examples:

```
notify_classes = bounce, delay, policy, protocol, resource, software
notify_classes = 2bounce, resource, software
```

owner_request_special (default: yes)

Give special treatment to owner-listname and listname-request address localparts: don't split such addresses when the recipient_delimiter is set to "-". This feature is useful for mailing lists.

parent_domain_matches_subdomains (default: see postconf -d output)

What Postfix features match subdomains of "domain.tld" automatically, instead of requiring an explicit ".domain.tld" pattern. This is planned backwards compatibility: eventually, all Postfix features are expected to require explicit ".domain.tld" style patterns when you really want to match subdomains.

permit_mx_backup_networks (default: empty)

Restrict the use of the permit_mx_backup SMTP access feature to only domains whose primary MX hosts match the listed networks.

pickup_service_name (default: pickup)

The name of the **pickup**(8) service. This service picks up local mail submissions from the Postfix maildrop queue.

This feature is available in Postfix 2.0 and later.

plaintext_reject_code (default: 450)
The numerical Postfix SMTP server response code when a request is rejected by the **reject_plaintext_session** restriction.

This feature is available in Postfix 2.3 and later.

prepend_delivered_header (default: command, file, forward)
The message delivery contexts where the Postfix **local**(8) delivery agent prepends a Delivered-To: message header with the address that the mail was delivered to. This information is used for mail delivery loop detection.

By default, the Postfix local delivery agent prepends a Delivered-To: header when forwarding mail and when delivering to file (mailbox) and command. Turning off the Delivered-To: header when forwarding mail is not recommended.

Specify zero or more of **forward**, **file**, or **command**.

Example:

```
prepend_delivered_header = forward
```

process_id (read-only)
The process ID of a Postfix command or daemon process.

process_id_directory (default: pid)
The location of Postfix PID files relative to $queue_directory. This is a read-only parameter.

process_name (read-only)
The process name of a Postfix command or daemon process.

propagate_unmatched_extensions (default: canonical, virtual)
What address lookup tables copy an address extension from the lookup key to the lookup result.

For example, with a **virtual**(5) mapping of "*joe@domain -> joe.user*", the address "*joe+foo@domain*" would rewrite to "*joe.user+foo*".

Specify zero or more of **canonical**, **virtual**, **alias**, **forward**, **include** or **generic**. These cause address extension propagation with **canonical**(5), **virtual**(5), and **aliases**(5) maps, with **local**(8) .forward and :include: file lookups, and with **smtp**(8) generic maps, respectively.

Note: enabling this feature for types other than **canonical** and **virtual** is likely to cause problems when mail is forwarded to other sites, especially with mail that is sent to a mailing list exploder address.

Examples:

```
propagate_unmatched_extensions = canonical, virtual, alias,
        forward, include
propagate_unmatched_extensions = canonical, virtual
```

proxy_interfaces (default: empty)
The network interface addresses that this mail system receives mail on by way of a proxy or network address translation unit.

This feature is available in Postfix 2.0 and later.

You must specify your "outside" proxy/NAT addresses when your system is a backup MX host for other domains, otherwise mail delivery loops will happen when the primary MX host is down.

Example:

```
proxy_interfaces = 1.2.3.4
```

proxy_read_maps (default: see postconf -d output)
The lookup tables that the **proxymap**(8) server is allowed to access. Table references that don’t begin with proxy: are ignored. The **proxymap**(8) table accesses are read-only.

This feature is available in Postfix 2.0 and later.

qmgr_clog_warn_time (default: 300s)
The minimal delay between warnings that a specific destination is clogging up the Postfix active queue. Specify 0 to disable.

This feature is enabled with the helpful_warnings parameter.

This feature is available in Postfix 2.0 and later.

qmgr_fudge_factor (default: 100)
Obsolete feature: the percentage of delivery resources that a busy mail system will use up for delivery of a large mailing list message.

This feature exists only in the **oqmgr**(8) old queue manager. The current queue manager solves the problem in a better way.

qmgr_message_active_limit (default: 20000)
The maximal number of messages in the active queue.

qmgr_message_recipient_limit (default: 20000)
The maximal number of recipients held in memory by the Postfix queue manager, and the maximal size of the size of the short-term, in-memory "dead" destination status cache.

qmgr_message_recipient_minimum (default: 10)
The minimal number of in-memory recipients for any message. This takes priority over any other in-memory recipient limits (i.e., the global qmgr_message_recipient_limit and the per transport _recipient_limit) if necessary. The minimum value allowed for this parameter is 1.

qmqpd_authorized_clients (default: empty)
What clients are allowed to connect to the QMQP server port.

By default, no client is allowed to use the service. This is because the QMQP server will relay mail to any destination.

Specify a list of client patterns. A list pattern specifies a host name, a domain name, an internet address, or a network/mask pattern, where the mask specifies the number of bits in the network part. When a pattern specifies a file name, its contents are substituted for the file name; when a pattern is a "type:table" table specification, table lookup is used instead.

Patterns are separated by whitespace and/or commas. In order to reverse the result, precede a pattern with an exclamation point (!). The form "!/file/name" is supported only in Postfix version 2.4 and later.

Example:

```
qmqpd_authorized_clients = !192.168.0.1, 192.168.0.0/24
```

qmqpd_error_delay (default: 1s)
How long the QMQP server will pause before sending a negative reply to the client. The purpose is to slow down confused or malicious clients.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

qmqpd_timeout (default: 300s)
The time limit for sending or receiving information over the network. If a read or write operation blocks for more than $qmqpd_timeout seconds the QMQP server gives up and disconnects.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

queue_directory (default: see postconf -d output)
The location of the Postfix top-level queue directory. This is the root directory of Postfix daemon processes that run chrooted.

queue_file_attribute_count_limit (default: 100)
The maximal number of (name=value) attributes that may be stored in a Postfix queue file. The limit is enforced by the **cleanup**(8) server.

This feature is available in Postfix 2.0 and later.

queue_minfree (default: 0)

The minimal amount of free space in bytes in the queue file system that is needed to receive mail. This is currently used by the SMTP server to decide if it will accept any mail at all.

By default, the Postfix version 2.1 SMTP server rejects MAIL FROM commands when the amount of free space is less than 1.5*$message_size_limit. To specify a higher minimum free space limit, specify a queue_minfree value that is at least 1.5*$message_size_limit.

With Postfix versions 2.0 and earlier, a queue_minfree value of zero means there is no minimum required amount of free space.

queue_run_delay (default: 300s)

The time between deferred queue scans by the queue manager; prior to Postfix 2.4 the default value was 1000s.

This parameter should be set less than or equal to $minimal_backoff_time. See also $maximal_backoff_time.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

queue_service_name (default: qmgr)

The name of the **qmgr**(8) service. This service manages the Postfix queue and schedules delivery requests.

This feature is available in Postfix 2.0 and later.

rbl_reply_maps (default: empty)

Optional lookup tables with RBL response templates. The tables are indexed by the RBL domain name. By default, Postfix uses the default template as specified with the default_rbl_reply configuration parameter. See there for a discussion of the syntax of RBL reply templates.

This feature is available in Postfix 2.0 and later.

readme_directory (default: see postconf -d output)

The location of Postfix README files that describe how to build, configure or operate a specific Postfix subsystem or feature.

receive_override_options (default: empty)

Enable or disable recipient validation, built-in content filtering, or address mapping. Typically, these are specified in master.cf as command-line arguments for the **smtpd**(8), **qmqpd**(8) or **pickup**(8) daemons.

Specify zero or more of the following options. The options override main.cf settings and are either implemented by **smtpd**(8), **qmqpd**(8), or **pickup**(8) themselves, or they are forwarded to the cleanup server.

no_unknown_recipient_checks

Do not try to reject unknown recipients (SMTP server only). This is typically specified AFTER an external content filter.

no_address_mappings

Disable canonical address mapping, virtual alias map expansion, address masquerading, and automatic BCC (blind carbon-copy) recipients. This is typically specified BEFORE an external content filter.

no_header_body_checks

Disable header/body_checks. This is typically specified AFTER an external content filter.

no_milters

Disable Milter (mail filter) applications. This is typically specified AFTER an external content filter.

Note: when the "BEFORE content filter" receive_override_options setting is specified in the main.cf file, specify the "AFTER content filter" receive_override_options setting in master.cf (and vice versa).

Examples:

```
receive_override_options =
    no_unknown_recipient_checks, no_header_body_checks
receive_override_options = no_address_mappings
```

This feature is available in Postfix 2.1 and later.

recipient_bcc_maps (default: empty)

Optional BCC (blind carbon-copy) address lookup tables, indexed by recipient address. The BCC address (multiple results are not supported) is added when mail enters from outside of Postfix.

This feature is available in Postfix 2.1 and later.

The table search order is as follows:

- Look up the "user+extension@domain.tld" address including the optional address extension.
- Look up the "user@domain.tld" address without the optional address extension.
- Look up the "user+extension" address local part when the recipient domain equals $myorigin, $mydestination, $inet_interfaces or $proxy_interfaces.
- Look up the "user" address local part when the recipient domain equals $myorigin, $mydestination, $inet_interfaces or $proxy_interfaces.
- Look up the "@domain.tld" part.

Specify the types and names of databases to use. After change, run "**postmap /etc/postfix/recipient_bcc**".

Note: if mail to the BCC address bounces it will be returned to the sender.

Note: automatic BCC recipients are produced only for new mail. To avoid mailer loops, automatic BCC recipients are not generated for mail that Postfix forwards internally, nor for mail that Postfix generates itself.

Example:

```
recipient_bcc_maps = hash:/etc/postfix/recipient_bcc
```

recipient_canonical_classes (default: envelope_recipient, header_recipient)

What addresses are subject to recipient_canonical_maps address mapping. By default, recipient_canonical_maps address mapping is applied to envelope recipient addresses, and to header recipient addresses.

Specify one or more of: envelope_recipient, header_recipient

This feature is available in Postfix 2.2 and later.

recipient_canonical_maps (default: empty)

Optional address mapping lookup tables for envelope and header recipient addresses. The table format and lookups are documented in **canonical**(5).

Note: $recipient_canonical_maps is processed before $canonical_maps.

Example:

```
recipient_canonical_maps = hash:/etc/postfix/recipient_canonical
```

recipient_delimiter (default: empty)

The separator between user names and address extensions (user+foo). See **canonical**(5), **local**(8), **relocated**(5) and **virtual**(5) for the effects this has on aliases, canonical, virtual, relocated and on .forward file lookups. Basically, the software tries user+foo and .forward+foo before trying user and .forward.

Example:

```
recipient_delimiter = +
```

reject_code (default: 554)

The numerical Postfix SMTP server response code when a remote SMTP client request is rejected by the "reject" restriction.

Do not change this unless you have a complete understanding of RFC 821.

relay_clientcerts (default: empty)

The list of remote SMTP client certificates for which the Postfix SMTP server will allow access with the permit_tls_clientcerts feature. This feature does not use certificate names, because Postfix list manipulation routines treat whitespace and some other characters as special. Instead we use certificate fingerprints as they are difficult to fake but easy to use for lookup.

Postfix lookup tables are in the form of (key, value) pairs. Since we only need the key, the value can be chosen freely, e.g. the name of the user or host: D7:04:2F:A7:0B:8C:A5:21:FA:31:77:E1:41:8A:EE:80 lutzpc.at.home

Example:

```
relay_clientcerts = hash:/etc/postfix/relay_clientcerts
```

For more fine-grained control, use check_ccert_access to select an appropriate **access**(5) policy for each client. See RESTRICTION_CLASS_README.

This feature is available with Postfix version 2.2.

relay_destination_concurrency_limit (default: $default_destination_concurrency_limit)

The maximal number of parallel deliveries to the same destination via the relay message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

This feature is available in Postfix 2.0 and later.

relay_destination_recipient_limit (default: $default_destination_recipient_limit)

The maximal number of recipients per delivery via the relay message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

Setting this parameter to a value of 1 changes the meaning of relay_destination_concurrency_limit from concurrency per domain into concurrency per recipient.

This feature is available in Postfix 2.0 and later.

relay_domains (default: $mydestination)

What destination domains (and subdomains thereof) this system will relay mail to. Subdomain matching is controlled with the parent_domain_matches_subdomains parameter. For details about how the relay_domains value is used, see the description of the permit_auth_destination and reject_unauth_destination SMTP recipient restrictions.

Domains that match $relay_domains are delivered with the $relay_transport mail delivery transport. The SMTP server validates recipient addresses with $relay_recipient_maps and rejects non-existent recipients. See also the relay domains address class in the ADDRESS_CLASS_README file.

Note: Postfix will not automatically forward mail for domains that list this system as their primary or backup MX host. See the permit_mx_backup restriction in the **postconf**(5) manual page.

Specify a list of host or domain names, "/file/name" patterns or "type:table" lookup tables, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a (parent) domain appears as lookup key. Specify "!pattern" to exclude a domain from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

relay_domains_reject_code (default: 554)

The numerical Postfix SMTP server response code when a client request is rejected by the reject_unauth_destination recipient restriction.

Do not change this unless you have a complete understanding of RFC 821.

relay_recipient_maps (default: empty)

Optional lookup tables with all valid addresses in the domains that match $relay_domains. Specify @domain as a wild-card for domains that have no valid recipient list, and become a source of backscatter mail: Postfix accepts spam for non-existent recipients and then floods innocent people with undeliverable

mail. Technically, tables listed with $relay_recipient_maps are used as lists: Postfix needs to know only if a lookup string is found or not, but it does not use the result from table lookup.

If this parameter is non-empty, then the Postfix SMTP server will reject mail to unknown relay users. This feature is off by default.

See also the relay domains address class in the ADDRESS_CLASS_README file.

Example:

```
relay_recipient_maps = hash:/etc/postfix/relay_recipients
```

This feature is available in Postfix 2.0 and later.

relay_transport (default: relay)

The default mail delivery transport and next-hop destination for remote delivery to domains listed with $relay_domains. In order of decreasing precedence, the nexthop destination is taken from $relay_transport, $sender_dependent_relayhost_maps, $relayhost, or from the recipient domain. This information can be overruled with the **transport**(5) table.

Specify a string of the form *transport:nexthop*, where *transport* is the name of a mail delivery transport defined in master.cf. The *:nexthop* part is optional. For more details see the **transport**(5) manual page.

See also the relay domains address class in the ADDRESS_CLASS_README file.

This feature is available in Postfix 2.0 and later.

relayhost (default: empty)

The next-hop destination of non-local mail; overrides non-local domains in recipient addresses. This information is overruled with relay_transport, default_transport, sender_dependent_relayhost_maps and with the **transport**(5) table.

On an intranet, specify the organizational domain name. If your internal DNS uses no MX records, specify the name of the intranet gateway host instead.

In the case of SMTP, specify a domain name, hostname, hostname:port, [hostname]:port, [hostaddress] or [hostaddress]:port. The form [hostname] turns off MX lookups.

If you're connected via UUCP, see the UUCP_README file for useful information.

Examples:

```
relayhost = $mydomain
relayhost = [gateway.my.domain]
relayhost = uucphost
relayhost = [an.ip.add.ress]
```

relocated_maps (default: empty)

Optional lookup tables with new contact information for users or domains that no longer exist. The table format and lookups are documented in **relocated**(5).

If you use this feature, run "**postmap /etc/postfix/relocated**" to build the necessary DBM or DB file after change, then "**postfix reload**" to make the changes visible.

Examples:

```
relocated_maps = dbm:/etc/postfix/relocated
relocated_maps = hash:/etc/postfix/relocated
```

remote_header_rewrite_domain (default: empty)

Don't rewrite message headers from remote clients at all when this parameter is empty; otherwise, rewrite message headers and append the specified domain name to incomplete addresses. The local_header_rewrite_clients parameter controls what clients Postfix considers local.

Examples:

The safe setting: append "domain.invalid" to incomplete header addresses from remote SMTP clients, so that those addresses cannot be confused with local addresses.

```
remote_header_rewrite_domain = domain.invalid
```

The default, purist, setting: don't rewrite headers from remote clients at all.

```
remote_header_rewrite_domain =
```

require_home_directory (default: no)
Whether or not a **local**(8) recipient's home directory must exist before mail delivery is attempted. By default this test is disabled. It can be useful for environments that import home directories to the mail server (NOT RECOMMENDED).

resolve_dequoted_address (default: yes)
Resolve a recipient address safely instead of correctly, by looking inside quotes.

By default, the Postfix address resolver does not quote the address localpart as per RFC 822, so that additional @ or % or ! operators remain visible. This behavior is safe but it is also technically incorrect.

If you specify "resolve_dequoted_address = no", then the Postfix resolver will not know about additional @ etc. operators in the address localpart. This opens opportunities for obscure mail relay attacks with user@domain@domain addresses when Postfix provides backup MX service for Sendmail systems.

resolve_null_domain (default: no)
Resolve an address that ends in the "@" null domain as if the local hostname were specified, instead of rejecting the address as invalid.

This feature is available in Postfix 2.1 and later. Earlier versions always resolve the null domain as the local hostname.

The Postfix SMTP server uses this feature to reject mail from or to addresses that end in the "@" null domain, and from addresses that rewrite into a form that ends in the "@" null domain.

resolve_numeric_domain (default: no)
Resolve "user@ipaddress" as "user@[ipaddress]", instead of rejecting the address as invalid.

This feature is available in Postfix 2.3 and later.

rewrite_service_name (default: rewrite)
The name of the address rewriting service. This service rewrites addresses to standard form and resolves them to a (delivery method, next-hop host, recipient) triple.

This feature is available in Postfix 2.0 and later.

sample_directory (default: /etc/postfix)
The name of the directory with example Postfix configuration files.

send_cyrus_sasl_authzid (default: no)
When authenticating to a remote SMTP or LMTP server with the default setting "no", send no SASL authoriZation ID (authzid); send only the SASL authentiCation ID (authcid) plus the authcid's password.

The non-default setting "yes" enables the behavior of older Postfix versions. These always send a SASL authzid that is equal to the SASL authcid, but this causes inter-operability problems with some SMTP servers.

This feature is available in Postfix 2.4.4 and later.

sender_based_routing (default: no)
This parameter should not be used. It was replaced by sender_dependent_relayhost_maps in Postfix version 2.3.

sender_bcc_maps (default: empty)
Optional BCC (blind carbon-copy) address lookup tables, indexed by sender address. The BCC address (multiple results are not supported) is added when mail enters from outside of Postfix.

This feature is available in Postfix 2.1 and later.

The table search order is as follows:

- Look up the "user+extension@domain.tld" address including the optional address extension.
- Look up the "user@domain.tld" address without the optional address extension.
- Look up the "user+extension" address local part when the sender domain equals $myorigin, $mydestination, $inet_interfaces or $proxy_interfaces.
- Look up the "user" address local part when the sender domain equals $myorigin, $mydestination, $inet_interfaces or $proxy_interfaces.
- Look up the "@domain.tld" part.

Specify the types and names of databases to use. After change, run "**postmap /etc/postfix/sender_bcc**".

Note: if mail to the BCC address bounces it will be returned to the sender.

Note: automatic BCC recipients are produced only for new mail. To avoid mailer loops, automatic BCC recipients are not generated for mail that Postfix forwards internally, nor for mail that Postfix generates itself.

Example:

```
sender_bcc_maps = hash:/etc/postfix/sender_bcc
```

sender_canonical_classes (default: envelope_sender, header_sender)

What addresses are subject to sender_canonical_maps address mapping. By default, sender_canonical_maps address mapping is applied to envelope sender addresses, and to header sender addresses.

Specify one or more of: envelope_sender, header_sender

This feature is available in Postfix 2.2 and later.

sender_canonical_maps (default: empty)

Optional address mapping lookup tables for envelope and header sender addresses. The table format and lookups are documented in **canonical**(5).

Example: you want to rewrite the SENDER address "user@ugly.domain" to "user@pretty.domain", while still being able to send mail to the RECIPIENT address "user@ugly.domain".

Note: $sender_canonical_maps is processed before $canonical_maps.

Example:

```
sender_canonical_maps = hash:/etc/postfix/sender_canonical
```

sender_dependent_relayhost_maps (default: empty)

A sender-dependent override for the global relayhost parameter setting. The tables are searched by the envelope sender address and @domain. This information is overruled with relay_transport, default_transport and with the **transport**(5) table.

For safety reasons, this feature does not allow $number substitutions in regular expression maps.

This feature is available in Postfix 2.3 and later.

sendmail_path (default: see postconf -d output)

A Sendmail compatibility feature that specifies the location of the Postfix **sendmail**(1) command. This command can be used to submit mail into the Postfix queue.

service_throttle_time (default: 60s)

How long the Postfix **master**(8) waits before forking a server that appears to be malfunctioning.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

setgid_group (default: postdrop)

The group ownership of set-gid Postfix commands and of group-writable Postfix directories. When this parameter value is changed you need to re-run "**postfix set-permissions**" (with Postfix version 2.0 and earlier: "**/etc/postfix/post-install set-permissions**".

show_user_unknown_table_name (default: yes)
Display the name of the recipient table in the "User unknown" responses. The extra detail makes trouble shooting easier but also reveals information that is nobody elses business.

This feature is available in Postfix 2.0 and later.

showq_service_name (default: showq)
The name of the **showq**(8) service. This service produces mail queue status reports.

This feature is available in Postfix 2.0 and later.

smtp_always_send_ehlo (default: yes)
Always send EHLO at the start of an SMTP session.

With "smtp_always_send_ehlo = no", Postfix sends EHLO only when the word "ESMTP" appears in the server greeting banner (example: 220 spike.porcupine.org ESMTP Postfix).

smtp_bind_address (default: empty)
An optional numerical network address that the Postfix SMTP client should bind to when making an IPv4 connection.

This can be specified in the main.cf file for all SMTP clients, or it can be specified in the master.cf file for a specific client, for example:

```
/etc/postfix/master.cf:
      smtp ... smtp -o smtp_bind_address=11.22.33.44
```

Note 1: when inet_interfaces specifies no more than one IPv4 address, and that address is a non-loopback address, it is automatically used as the smtp_bind_address. This supports virtual IP hosting, but can be a problem on multi-homed firewalls. See the inet_interfaces documentation for more detail.

Note 2: address information may be enclosed inside [], but this form is not required here.

smtp_bind_address6 (default: empty)
An optional numerical network address that the Postfix SMTP client should bind to when making an IPv6 connection.

This feature is available in Postfix 2.2 and later.

This can be specified in the main.cf file for all SMTP clients, or it can be specified in the master.cf file for a specific client, for example:

```
/etc/postfix/master.cf:
      smtp ... smtp -o smtp_bind_address6=1:2:3:4:5:6:7:8
```

Note 1: when inet_interfaces specifies no more than one IPv6 address, and that address is a non-loopback address, it is automatically used as the smtp_bind_address6. This supports virtual IP hosting, but can be a problem on multi-homed firewalls. See the inet_interfaces documentation for more detail.

Note 2: address information may be enclosed inside [], but this form is not recommended here.

smtp_cname_overrides_servername (default: version dependent)
Allow DNS CNAME records to override the servername that the Postfix SMTP client uses for logging, SASL password lookup, TLS policy decisions, or TLS certificate verification. The value "no" hardens Postfix smtp_tls_per_site hostname-based policies against false hostname information in DNS CNAME records, and makes SASL password file lookups more predictable. This is the default setting as of Postfix 2.3.

This feature is available in Postfix 2.2.9 and later.

smtp_connect_timeout (default: 30s)
The SMTP client time limit for completing a TCP connection, or zero (use the operating system built-in time limit).

When no connection can be made within the deadline, the Postfix SMTP client tries the next address on the mail exchanger list. Specify 0 to disable the time limit (i.e. use whatever timeout is implemented by the

operating system).

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_connection_cache_destinations (default: empty)

Permanently enable SMTP connection caching for the specified destinations. With SMTP connection caching, a connection is not closed immediately after completion of a mail transaction. Instead, the connection is kept open for up to $smtp_connection_cache_time_limit seconds. This allows connections to be reused for other deliveries, and can improve mail delivery performance.

Specify a comma or white space separated list of destinations or pseudo-destinations:

- if mail is sent without a relay host: a domain name (the right-hand side of an email address, without the [] around a numeric IP address),
- if mail is sent via a relay host: a relay host name (without [] or non-default TCP port), as specified in main.cf or in the transport map,
- if mail is sent via a UNIX-domain socket: a pathname (without the unix: prefix),
- a /file/name with domain names and/or relay host names as defined above,
- a "type:table" with domain names and/or relay hosts name on the left-hand side. The right-hand side result from "type:table" lookups is ignored.

This feature is available in Postfix 2.2 and later.

smtp_connection_cache_on_demand (default: yes)

Temporarily enable SMTP connection caching while a destination has a high volume of mail in the active queue. With SMTP connection caching, a connection is not closed immediately after completion of a mail transaction. Instead, the connection is kept open for up to $smtp_connection_cache_time_limit seconds. This allows connections to be reused for other deliveries, and can improve mail delivery performance.

This feature is available in Postfix 2.2 and later.

smtp_connection_cache_reuse_limit (default: 10)

When SMTP connection caching is enabled, the number of times that an SMTP session may be reused before it is closed.

This feature is available in Postfix 2.2. In Postfix 2.3 it is replaced by $smtp_connection_reuse_time_limit.

smtp_connection_cache_time_limit (default: 2s)

When SMTP connection caching is enabled, the amount of time that an unused SMTP client socket is kept open before it is closed. Do not specify larger values without permission from the remote sites.

This feature is available in Postfix 2.2 and later.

smtp_connection_reuse_time_limit (default: 300s)

The amount of time during which Postfix will use an SMTP connection repeatedly. The timer starts when the connection is initiated (i.e. it includes the connect, greeting and helo latency, in addition to the latencies of subsequent mail delivery transactions).

This feature addresses a performance stability problem with remote SMTP servers. This problem is not specific to Postfix: it can happen when any MTA sends large amounts of SMTP email to a site that has multiple MX hosts.

The problem starts when one of a set of MX hosts becomes slower than the rest. Even though SMTP clients connect to fast and slow MX hosts with equal probability, the slow MX host ends up with more simultaneous inbound connections than the faster MX hosts, because the slow MX host needs more time to serve each client request.

The slow MX host becomes a connection attractor. If one MX host becomes N times slower than the rest, it dominates mail delivery latency unless there are more than N fast MX hosts to counter the effect. And if the number of MX hosts is smaller than N, the mail delivery latency becomes effectively that of the slowest MX host divided by the total number of MX hosts.

The solution uses connection caching in a way that differs from Postfix version 2.2. By limiting the amount of time during which a connection can be used repeatedly (instead of limiting the number of deliveries over that connection), Postfix not only restores fairness in the distribution of simultaneous connections across a set of MX hosts, it also favors deliveries over connections that perform well, which is exactly what we want.

The default reuse time limit, 300s, is comparable to the various smtp transaction timeouts which are fair estimates of maximum excess latency for a slow delivery. Note that hosts may accept thousands of messages over a single connection within the default connection reuse time limit. This number is much larger than the default Postfix version 2.2 limit of 10 messages per cached connection. It may prove necessary to lower the limit to avoid interoperability issues with MTAs that exhibit bugs when many messages are delivered via a single connection. A lower reuse time limit risks losing the benefit of connection reuse when the average connection and mail delivery latency exceeds the reuse time limit.

This feature is available in Postfix 2.3 and later.

smtp_data_done_timeout (default: 600s)

The SMTP client time limit for sending the SMTP ".", and for receiving the server response.

When no response is received within the deadline, a warning is logged that the mail may be delivered multiple times.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_data_init_timeout (default: 120s)

The SMTP client time limit for sending the SMTP DATA command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_data_xfer_timeout (default: 180s)

The SMTP client time limit for sending the SMTP message content. When the connection makes no progress for more than $smtp_data_xfer_timeout seconds the Postfix SMTP client terminates the transfer.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_defer_if_no_mx_address_found (default: no)

Defer mail delivery when no MX record resolves to an IP address.

The default (no) is to return the mail as undeliverable. With older Postfix versions the default was to keep trying to deliver the mail until someone fixed the MX record or until the mail was too old.

Note: Postfix always ignores MX records with equal or worse preference than the local MTA itself.

This feature is available in Postfix 2.1 and later.

smtp_destination_concurrency_limit (default: $default_destination_concurrency_limit)

The maximal number of parallel deliveries to the same destination via the smtp message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

smtp_destination_recipient_limit (default: $default_destination_recipient_limit)

The maximal number of recipients per delivery via the smtp message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

Setting this parameter to a value of 1 changes the meaning of smtp_destination_concurrency_limit from concurrency per domain into concurrency per recipient.

smtp_discard_ehlo_keyword_address_maps (default: empty)

Lookup tables, indexed by the remote SMTP server address, with case insensitive lists of EHLO keywords (pipelining, starttls, auth, etc.) that the Postfix SMTP client will ignore in the EHLO response from a remote SMTP server. See smtp_discard_ehlo_keywords for details. The table is not indexed by hostname for consistency with smtpd_discard_ehlo_keyword_address_maps.

This feature is available in Postfix 2.2 and later.

smtp_discard_ehlo_keywords (default: empty)

A case insensitive list of EHLO keywords (pipelining, starttls, auth, etc.) that the Postfix SMTP client will ignore in the EHLO response from a remote SMTP server.

This feature is available in Postfix 2.2 and later.

Notes:

- Specify the **silent-discard** pseudo keyword to prevent this action from being logged.
- Use the smtp_discard_ehlo_keyword_address_maps feature to discard EHLO keywords selectively.

smtp_enforce_tls (default: no)

Enforcement mode: require that remote SMTP servers use TLS encryption, and never send mail in the clear. This also requires that the remote SMTP server hostname matches the information in the remote server certificate, and that the remote SMTP server certificate was issued by a CA that is trusted by the Postfix SMTP client. If the certificate doesn't verify or the hostname doesn't match, delivery is deferred and mail stays in the queue.

The server hostname is matched against all names provided as dNSNames in the SubjectAlternativeName. If no dNSNames are specified, the CommonName is checked. The behavior may be changed with the smtp_tls_enforce_peername option.

This option is useful only if you are definitely sure that you will only connect to servers that support RFC 2487 _and_ that provide valid server certificates. Typical use is for clients that send all their email to a dedicated mailhub.

This feature is available in Postfix 2.2 and later. With Postfix 2.3 and later use smtp_tls_security_level instead.

smtp_fallback_relay (default: $fallback_relay)

Optional list of relay hosts for SMTP destinations that can't be found or that are unreachable. With Postfix 2.2 and earlier this parameter is called fallback_relay.

By default, mail is returned to the sender when a destination is not found, and delivery is deferred when a destination is unreachable.

The fallback relays must be SMTP destinations. Specify a domain, host, host:port, [host]:port, [address] or [address]:port; the form [host] turns off MX lookups. If you specify multiple SMTP destinations, Postfix will try them in the specified order.

To prevent mailer loops between MX hosts and fall-back hosts, Postfix version 2.3 and later will not use the smtp_fallback_relay feature for destinations that it is MX host for.

smtp_generic_maps (default: empty)

Optional lookup tables that perform address rewriting in the SMTP client, typically to transform a locally valid address into a globally valid address when sending mail across the Internet. This is needed when the local machine does not have its own Internet domain name, but uses something like *localdomain.local* instead.

The table format and lookups are documented in **generic**(5); examples are shown in the ADDRESS_REWRITING_README and STANDARD_CONFIGURATION_README documents.

This feature is available in Postfix 2.2 and later.

smtp_helo_name (default: $myhostname)

The hostname to send in the SMTP EHLO or HELO command.

The default value is the machine hostname. Specify a hostname or [ip.add.re.ss].

This information can be specified in the main.cf file for all SMTP clients, or it can be specified in the master.cf file for a specific client, for example:

```
/etc/postfix/master.cf:
      mysmtp ... smtp -o smtp_helo_name=foo.bar.com
```

This feature is available in Postfix 2.0 and later.

smtp_helo_timeout (default: 300s)
The SMTP client time limit for sending the HELO or EHLO command, and for receiving the initial server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_host_lookup (default: dns)
What mechanisms when the Postfix SMTP client uses to look up a host's IP address. This parameter is ignored when DNS lookups are disabled.

Specify one of the following:

dns Hosts can be found in the DNS (preferred).

native Use the native naming service only (nsswitch.conf, or equivalent mechanism).

dns, native
Use the native service for hosts not found in the DNS.

This feature is available in Postfix 2.1 and later.

smtp_line_length_limit (default: 990)
The maximal length of message header and body lines that Postfix will send via SMTP. Longer lines are broken by inserting "<CR><LF><SPACE>". This minimizes the damage to MIME formatted mail.

By default, the line length is limited to 990 characters, because some server implementations cannot receive mail with long lines.

smtp_mail_timeout (default: 300s)
The SMTP client time limit for sending the MAIL FROM command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_mx_address_limit (default: 5)
The maximal number of MX (mail exchanger) IP addresses that can result from mail exchanger lookups, or zero (no limit). Prior to Postfix version 2.3, this limit was disabled by default.

This feature is available in Postfix 2.1 and later.

smtp_mx_session_limit (default: 2)
The maximal number of SMTP sessions per delivery request before giving up or delivering to a fall-back relay host, or zero (no limit). This restriction ignores sessions that fail to complete the SMTP initial handshake (Postfix version 2.2 and earlier) or that fail to complete the EHLO and TLS handshake (Postfix version 2.3 and later).

This feature is available in Postfix 2.1 and later.

smtp_never_send_ehlo (default: no)
Never send EHLO at the start of an SMTP session. See also the smtp_always_send_ehlo parameter.

smtp_pix_workaround_delay_time (default: 10s)
How long the Postfix SMTP client pauses before sending ".<CR><LF>" in order to work around the PIX firewall "<CR><LF>.<CR><LF>" bug.

Choosing a too short time makes this workaround ineffective when sending large messages over slow network connections.

smtp_pix_workaround_maps (default: empty)
Lookup tables, indexed by the remote SMTP server address, with per-destination workarounds for CISCO PIX firewall bugs. The table is not indexed by hostname for consistency with smtp_discard_ehlo_keyword_address_maps.

This feature is available in Postfix 2.4 and later.

smtp_pix_workaround_threshold_time (default: 500s)
How long a message must be queued before the Postfix SMTP client turns on the PIX firewall "<CR><LF>.<CR><LF>" bug workaround for delivery through firewalls with "smtp fixup" mode turned on.

By default, the workaround is turned off for mail that is queued for less than 500 seconds. In other words, the workaround is normally turned off for the first delivery attempt.

Specify 0 to enable the PIX firewall "<CR><LF>.<CR><LF>" bug workaround upon the first delivery attempt.

smtp_pix_workarounds (default: disable_esmtp, delay_dotcrlf)
A list that specifies zero or more workarounds for CISCO PIX firewall bugs. These workarounds are implemented by the Postfix SMTP client. Workaround names are separated by comma or space, and are case insensitive. This parameter setting can be overruled with per-destination smtp_pix_workaround_maps settings.

delay_dotcrlf
Insert a delay before sending ".<CR><LF>" after the end of the message content. The delay is subject to the smtp_pix_workaround_delay_time and smtp_pix_workaround_threshold_time parameter settings.

disable_esmtp
Disable all extended SMTP commands: send HELO instead of EHLO.

This feature is available in Postfix 2.4 and later. The default settings are backwards compatible with earlier Postfix versions.

smtp_quit_timeout (default: 300s)
The SMTP client time limit for sending the QUIT command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_quote_rfc821_envelope (default: yes)
Quote addresses in SMTP MAIL FROM and RCPT TO commands as required by RFC 821. This includes putting quotes around an address localpart that ends in ".".

The default is to comply with RFC 821. If you have to send mail to a broken SMTP server, configure a special SMTP client in master.cf:

```
/etc/postfix/master.cf:
    broken-smtp . . . smtp -o smtp_quote_rfc821_envelope=no
```

and route mail for the destination in question to the "broken-smtp" message delivery with a **transport**(5) table.

This feature is available in Postfix 2.1 and later.

smtp_randomize_addresses (default: yes)
Randomize the order of equal-preference MX host addresses. This is a performance feature of the Postfix SMTP client.

smtp_rcpt_timeout (default: 300s)
The SMTP client time limit for sending the SMTP RCPT TO command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtp_rset_timeout (default: 20s)
The SMTP client time limit for sending the RSET command, and for receiving the server response. The SMTP client sends RSET in order to finish a recipient address probe, or to verify that a cached session is still usable.

This feature is available in Postfix 2.1 and later.

smtp_sasl_auth_enable (default: no)

Enable SASL authentication in the Postfix SMTP client. By default, the Postfix SMTP client uses no authentication.

Example:

```
smtp_sasl_auth_enable = yes
```

smtp_sasl_mechanism_filter (default: empty)

If non-empty, a Postfix SMTP client filter for the remote SMTP server's list of offered SASL mechanisms. Different client and server implementations may support different mechanism lists. By default, the Postfix SMTP client will use the intersection of the two. smtp_sasl_mechanism_filter further restricts what server mechanisms the client will take into consideration.

Specify mechanism names, "/file/name" patterns or "type:table" lookup tables. The right-hand side result from "type:table" lookups is ignored. Specify "!pattern" to exclude a mechanism name from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

This feature is available in Postfix 2.2 and later.

Examples:

```
smtp_sasl_mechanism_filter = plain, login
smtp_sasl_mechanism_filter = /etc/postfix/smtp_mechs
smtp_sasl_mechanism_filter = !gssapi, !login, static:rest
```

smtp_sasl_password_maps (default: empty)

Optional SMTP client lookup tables with one username:password entry per remote hostname or domain, or sender address when sender-dependent authentication is enabled. If no username:password entry is found, then the Postfix SMTP client will not attempt to authenticate to the remote host.

The Postfix SMTP client opens the lookup table before going to chroot jail, so you can leave the password file in /etc/postfix.

smtp_sasl_path (default: empty)

Implementation-specific information that is passed through to the SASL plug-in implementation that is selected with **smtp_sasl_type**. Typically this specifies the name of a configuration file or rendezvous point.

This feature is available in Postfix 2.3 and later.

smtp_sasl_security_options (default: noplaintext, noanonymous)

SASL security options; as of Postfix 2.3 the list of available features depends on the SASL client implementation that is selected with **smtp_sasl_type**.

The following security features are defined for the **cyrus** client SASL implementation:

Specify zero or more of the following:

noplaintext
: Disallow methods that use plaintext passwords.

noactive
: Disallow methods subject to active (non-dictionary) attack.

nodictionary
: Disallow methods subject to passive (dictionary) attack.

noanonymous
: Disallow methods that allow anonymous authentication.

mutual_auth
: Only allow methods that provide mutual authentication (not available with SASL version 1).

Example:

```
smtp_sasl_security_options = noplaintext
```

smtp_sasl_tls_security_options (default: $smtp_sasl_security_options)
The SASL authentication security options that the Postfix SMTP client uses for TLS encrypted SMTP sessions.

This feature is available in Postfix 2.2 and later.

smtp_sasl_tls_verified_security_options (default: $smtp_sasl_tls_security_options)
The SASL authentication security options that the Postfix SMTP client uses for TLS encrypted SMTP sessions with a verified server certificate. This feature is still under construction. It will not be included in the Postfix 2.3 release.

This feature should be available in Postfix 2.4 and later.

smtp_sasl_type (default: cyrus)
The SASL plug-in type that the Postfix SMTP client should use for authentication. The available types are listed with the "**postconf -A**" command.

This feature is available in Postfix 2.3 and later.

smtp_send_xforward_command (default: no)
Send the non-standard XFORWARD command when the Postfix SMTP server EHLO response announces XFORWARD support.

This allows an "smtp" delivery agent, used for injecting mail into a content filter, to forward the name, address, protocol and HELO name of the original client to the content filter and downstream queuing SMTP server. This can produce more useful logging than localhost[127.0.0.1] etc.

This feature is available in Postfix 2.1 and later.

smtp_sender_dependent_authentication (default: no)
Enable sender-dependent authentication in the Postfix SMTP client; this is available only with SASL authentication, and disables SMTP connection caching to ensure that mail from different senders will use the appropriate credentials.

This feature is available in Postfix 2.3 and later.

smtp_skip_4xx_greeting (default: yes)
Skip SMTP servers that greet with a 4XX status code (go away, try again later).

By default, Postfix moves on the next mail exchanger. Specify "smtp_skip_4xx_greeting = no" if Postfix should defer delivery immediately.

This feature is available in Postfix 2.0 and earlier. Later Postfix versions always skip SMTP servers that greet with a 4XX status code.

smtp_skip_5xx_greeting (default: yes)
Skip SMTP servers that greet with a 5XX status code (go away, do not try again later).

By default, the Postfix SMTP client moves on the next mail exchanger. Specify "smtp_skip_5xx_greeting = no" if Postfix should bounce the mail immediately. The default setting is incorrect, but it is what a lot of people expect to happen.

smtp_skip_quit_response (default: yes)
Do not wait for the response to the SMTP QUIT command.

smtp_starttls_timeout (default: 300s)
Time limit for Postfix SMTP client write and read operations during TLS startup and shutdown handshake procedures.

This feature is available in Postfix 2.2 and later.

smtp_tls_CAfile (default: empty)
The file with the certificate of the certification authority (CA) that issued the Postfix SMTP client certificate. This is needed only when the CA certificate is not already present in the client certificate file.

Example:

```
smtp_tls_CAfile = /etc/postfix/CAcert.pem
```

This feature is available in Postfix 2.2 and later.

smtp_tls_CApath (default: empty)

Directory with PEM format certificate authority certificates that the Postfix SMTP client uses to verify a remote SMTP server certificate. Don't forget to create the necessary "hash" links with, for example, "$OPENSSL_HOME/bin/c_rehash /etc/postfix/certs".

To use this option in chroot mode, this directory (or a copy) must be inside the chroot jail.

Example:

```
smtp_tls_CApath = /etc/postfix/certs
```

This feature is available in Postfix 2.2 and later.

smtp_tls_cert_file (default: empty)

File with the Postfix SMTP client RSA certificate in PEM format. This file may also contain the Postfix SMTP client private RSA key, and these may be the same as the Postfix SMTP server RSA certificate and key file.

Do not configure client certificates unless you **must** present client TLS certificates to one or more servers. Client certificates are not usually needed, and can cause problems in configurations that work well without them. The recommended setting is to let the defaults stand:

```
smtp_tls_cert_file =
smtp_tls_dcert_file =
smtp_tls_key_file =
smtp_tls_dkey_file =
```

The best way to use the default settings is to comment out the above parameters in main.cf if present.

In order to verify certificates, the CA certificate (in case of a certificate chain, all CA certificates) must be available. You should add these certificates to the client certificate, the client certificate first, then the issuing CA(s).

Example: the certificate for "client.dom.ain" was issued by "intermediate CA" which itself has a certificate of "root CA". Create the client.pem file with "cat client_cert.pem intermediate_CA.pem root_CA.pem > client.pem".

If you also want to verify remote SMTP server certificates issued by these CAs, you can also add the CA certificates to the smtp_tls_CAfile, in which case it is not necessary to have them in the smtp_tls_cert_file or smtp_tls_dcert_file.

A certificate supplied here must be usable as an SSL client certificate and hence pass the "openssl verify -purpose sslclient ..." test.

Example:

```
smtp_tls_cert_file = /etc/postfix/client.pem
```

This feature is available in Postfix 2.2 and later.

smtp_tls_cipherlist (default: empty)

Obsolete Postfix < 2.3 control for the Postfix SMTP client TLS cipher list. As this feature applies to all TLS security levels, it is easy to create inter-operability problems by choosing a non-default cipher list. Do not use a non-default TLS cipher list on hosts that deliver email to the public Internet: you will be unable to send email to servers that only support the ciphers you exclude. Using a restricted cipher list may be more appropriate for an internal MTA, where one can exert some control over the TLS software and settings of the peer servers.

Note: do not use "" quotes around the parameter value.

This feature is available in Postfix version 2.2. It is not used with Postfix 2.3 and later; use smtp_tls_mandatory_ciphers instead.

smtp_tls_dcert_file (default: empty)
File with the Postfix SMTP client DSA certificate in PEM format. This file may also contain the Postfix SMTP client private DSA key.

See the discussion under smtp_tls_cert_file for more details.

Example:

```
smtp_tls_dcert_file = /etc/postfix/client-dsa.pem
```

This feature is available in Postfix 2.2 and later.

smtp_tls_dkey_file (default: $smtp_tls_dcert_file)
File with the Postfix SMTP client DSA private key in PEM format. This file may be combined with the Postfix SMTP client DSA certificate file specified with $smtp_tls_dcert_file.

The private key must be accessible without a pass-phrase, i.e. it must not be encrypted, but file permissions should grant read/write access only to the system superuser account ("root").

This feature is available in Postfix 2.2 and later.

smtp_tls_enforce_peername (default: yes)
With mandatory TLS encryption, require that the remote SMTP server hostname matches the information in the remote SMTP server certificate. As of RFC 2487 the requirements for hostname checking for MTA clients are not specified.

This option can be set to "no" to disable strict peer name checking. This setting has no effect on sessions that are controlled via the smtp_tls_per_site table.

Disabling the hostname verification can make sense in closed environment where special CAs are created. If not used carefully, this option opens the danger of a "man-in-the-middle" attack (the CommonName of this attacker will be logged).

This feature is available in Postfix 2.2 and later. With Postfix 2.3 and later use smtp_tls_security_level instead.

smtp_tls_exclude_ciphers (default: empty)
List of ciphers or cipher types to exclude from the Postfix SMTP client cipher list at all TLS security levels. This is not an OpenSSL cipherlist, it is a simple list separated by whitespace and/or commas. The elements are a single cipher, or one or more "+" separated cipher properties, in which case only ciphers matching **all** the properties are excluded.

Examples (some of these will cause problems):

```
smtp_tls_exclude_ciphers = aNULL
smtp_tls_exclude_ciphers = MD5, DES
smtp_tls_exclude_ciphers = DES+MD5
smtp_tls_exclude_ciphers = AES256-SHA, DES-CBC3-MD5
smtp_tls_exclude_ciphers = kEDH+aRSA
```

The first setting, disables anonymous ciphers. The next setting disables ciphers that use the MD5 digest algorithm or the (single) DES encryption algorithm. The next setting disables ciphers that use MD5 and DES together. The next setting disables the two ciphers "AES256-SHA" and "DES-CBC3-MD5". The last setting disables ciphers that use "EDH" key exchange with RSA authentication.

This feature is available in Postfix 2.3 and later.

smtp_tls_key_file (default: $smtp_tls_cert_file)
File with the Postfix SMTP client RSA private key in PEM format. This file may be combined with the Postfix SMTP client RSA certificate file specified with $smtp_tls_cert_file.

The private key must be accessible without a pass-phrase, i.e. it must not be encrypted, but file permissions should grant read/write access only to the system superuser account ("root").

Example:

```
smtp_tls_key_file = $smtp_tls_cert_file
```

This feature is available in Postfix 2.2 and later.

smtp_tls_loglevel (default: 0)

Enable additional Postfix SMTP client logging of TLS activity. Each logging level also includes the information that is logged at a lower logging level.

0 Disable logging of TLS activity.

1 Log TLS handshake and certificate information.

2 Log levels during TLS negotiation.

3 Log hexadecimal and ASCII dump of TLS negotiation process.

4 Log hexadecimal and ASCII dump of complete transmission after STARTTLS.

Use "smtp_tls_loglevel = 3" only in case of problems. Use of loglevel 4 is strongly discouraged.

This feature is available in Postfix 2.2 and later.

smtp_tls_mandatory_ciphers (default: medium)

The minimum TLS cipher grade that the Postfix SMTP client will use with mandatory TLS encryption. The default value "medium" is suitable for most destinations with which you may want to enforce TLS, and is beyond the reach of today's crypt-analytic methods. See smtp_tls_policy_maps for information on how to configure ciphers on a per-destination basis.

The following cipher grades are supported:

export Enable the mainstream "EXPORT" grade or better OpenSSL ciphers. This is always used for opportunistic encryption. It is not recommended for mandatory encryption unless you must enforce TLS with "crippled" peers. The underlying cipherlist is specified via the tls_export_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_export_cipherlist includes anonymous ciphers, but these are automatically filtered out if the client is configured to verify server certificates. If you must exclude anonymous ciphers also at the "encrypt" security level, set "smtp_tls_mandatory_exclude_ciphers = aNULL".

low Enable the mainstream "LOW" grade or better OpenSSL ciphers. This setting is only appropriate for internal mail servers. The underlying cipherlist is specified via the tls_low_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_low_cipherlist includes anonymous ciphers, but these are automatically filtered out if the client is configured to verify server certificates. If you must exclude anonymous ciphers also at the "encrypt" security level, set "smtp_tls_mandatory_exclude_ciphers = aNULL".

medium
Enable the mainstream "MEDIUM" grade or better OpenSSL ciphers. The underlying cipherlist is specified via the tls_medium_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_medium_cipherlist includes anonymous ciphers, but these are automatically filtered out if the client is configured to verify server certificates. If you must exclude anonymous ciphers also at the "encrypt" security level, set "smtp_tls_mandatory_exclude_ciphers = aNULL".

high Enable only the mainstream "HIGH" grade OpenSSL ciphers. This setting is appropriate when all mandatory TLS destinations support some of "HIGH" grade ciphers, this is not uncommon. The underlying cipherlist is specified via the tls_high_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_high_cipherlist includes anonymous ciphers, but these are automatically filtered out if the client is configured to verify server certificates. If you must exclude anonymous ciphers also at the "encrypt" security level, set "smtp_tls_mandatory_exclude_ciphers = aNULL".

null Enable only the "NULL" OpenSSL ciphers, these provide authentication without encryption. This setting is only appropriate in the rare case that all servers are prepared to use NULL ciphers (not normally enabled in TLS servers). A plausible use-case is an LMTP server listening on a UNIX-

domain socket that is configured to support "NULL" ciphers. The underlying cipherlist is specified via the tls_null_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_null_cipherlist excludes anonymous ciphers (OpenSSL 0.9.8 has NULL ciphers that offer data integrity without encryption or authentication).

This feature is available in Postfix 2.3 and later.

smtp_tls_mandatory_exclude_ciphers (default: empty)

Additional list of ciphers or cipher types to exclude from the SMTP client cipher list at mandatory TLS security levels. This list works in addition to the exclusions listed with smtp_tls_exclude_ciphers (see there for syntax details).

This feature is available in Postfix 2.3 and later.

smtp_tls_mandatory_protocols (default: SSLv3, TLSv1)

List of TLS protocols that the Postfix SMTP client will use with mandatory TLS encryption. In main.cf the values are separated by whitespace, commas or colons. In the policy table (see smtp_tls_policy_maps) the only valid separator is colon. An empty value means allow all protocols. The valid protocol names, (see \fBfBSSL_get_version(3)), are "SSLv2", "SSLv3" and "TLSv1".

Since SSL version 2 has known protocol weaknesses and is now deprecated, the default setting only lists "SSLv3" and "TLSv1". This means that by default, SSL version 2 will not be used at the "encrypt" security level and higher.

See the documentation of the smtp_tls_policy_maps parameter and TLS_README for more information about security levels.

This feature is available in Postfix 2.3 and later.

smtp_tls_note_starttls_offer (default: no)

Log the hostname of a remote SMTP server that offers STARTTLS, when TLS is not already enabled for that server.

The logfile record looks like:

```
postfix/smtp[pid]:  Host offered STARTTLS: [name.of.host]
```

This feature is available in Postfix 2.2 and later.

smtp_tls_per_site (default: empty)

Optional lookup tables with the Postfix SMTP client TLS usage policy by next-hop destination and by remote SMTP server hostname. When both lookups succeed, the more specific per-site policy (NONE, MUST, etc) overrides the less specific one (MAY), and the more secure per-site policy (MUST, etc) overrides the less secure one (NONE). With Postfix 2.3 and later smtp_tls_per_site is strongly discouraged: use smtp_tls_policy_maps instead.

Use of the bare hostname as the per-site table lookup key is discouraged. Always use the full destination nexthop (enclosed in [] with a possible ":port" suffix). A recipient domain or MX-enabled transport nexthop with no port suffix may look like a bare hostname, but is still a suitable *destination*.

Specify a next-hop destination or server hostname on the left-hand side; no wildcards are allowed. The next-hop destination is either the recipient domain, or the destination specified with a **transport**(5) table, the relayhost parameter, or the relay_transport parameter. On the right hand side specify one of the following keywords:

NONE Don’t use TLS at all. This overrides a less specific **MAY** lookup result from the alternate host or next-hop lookup key, and overrides the global smtp_use_tls, smtp_enforce_tls, and smtp_tls_enforce_peername settings.

MAY Try to use TLS if the server announces support, otherwise use the unencrypted connection. This has less precedence than a more specific result (including **NONE**) from the alternate host or next-hop lookup key, and has less precedence than the more specific global "smtp_enforce_tls = yes" or "smtp_tls_enforce_peername = yes".

MUST_NOPEERMATCH
: Require TLS encryption, but do not require that the remote SMTP server hostname matches the information in the remote SMTP server certificate, or that the server certificate was issued by a trusted CA. This overrides a less secure **NONE** or a less specific **MAY** lookup result from the alternate host or next-hop lookup key, and overrides the global smtp_use_tls, smtp_enforce_tls and smtp_tls_enforce_peername settings.

MUST
: Require TLS encryption, require that the remote SMTP server hostname matches the information in the remote SMTP server certificate, and require that the remote SMTP server certificate was issued by a trusted CA. This overrides a less secure **NONE** and **MUST_NOPEERMATCH** or a less specific **MAY** lookup result from the alternate host or next-hop lookup key, and overrides the global smtp_use_tls, smtp_enforce_tls and smtp_tls_enforce_peername settings.

The above keywords correspond to the "none", "may", "encrypt" and "verify" security levels for the new smtp_tls_security_level parameter introduced in Postfix 2.3. Starting with Postfix 2.3, and independently of how the policy is specified, the smtp_tls_mandatory_ciphers and smtp_tls_mandatory_protocols parameters only apply when TLS encryption is mandatory. Connections for which encryption is optional enable all "export" grade and better ciphers.

As long as no secure DNS lookup mechanism is available, false hostnames in MX or CNAME responses can change the server hostname that Postfix uses for TLS policy lookup and server certificate verification. Even with a perfect match between the server hostname and the server certificate, there is no guarantee that Postfix is connected to the right server. See TLS_README (Closing a DNS loophole with obsolete per-site TLS policies) for a possible work-around.

This feature is available in Postfix 2.2 and later. With Postfix 2.3 and later use smtp_tls_policy_maps instead.

smtp_tls_policy_maps (default: empty)

Optional lookup tables with the Postfix SMTP client TLS security policy by next-hop destination; when a non-empty value is specified, this overrides the obsolete smtp_tls_per_site parameter. See TLS_README for a more detailed discussion of TLS security levels.

The TLS policy table is indexed by the full next-hop destination, which is either the recipient domain, or the verbatim next-hop specified in the transport table, $local_transport, $virtual_transport, $relay_transport or $default_transport. This includes any enclosing square brackets and any non-default destination server port suffix. The LMTP socket type prefix (inet: or unix:) is not included in the lookup key.

Only the next-hop domain, or $myhostname with LMTP over UNIX-domain sockets, is used as the nexthop name for certificate verification. The port and any enclosing square brackets are used in the table lookup key, but are not used for server name verification.

When the lookup key is a domain name without enclosing square brackets or any *:port* suffix (typically the recipient domain), and the full domain is not found in the table, just as with the **transport**(5) table, the parent domain starting with a leading "." is matched recursively. This allows one to specify a security policy for a recipient domain and all its sub-domains.

The lookup result is a security level, followed by an optional list of whitespace and/or comma separated name=value attributes that override related main.cf settings. The TLS security levels in order of increasing security are:

none
: No TLS. No additional attributes are supported at this level.

may
: Opportunistic TLS. No additional attributes are supported at this level. Since sending in the clear is acceptable, demanding stronger than default TLS security parameters merely reduces inter-operability. Postfix 2.3 and later ignore the smtp_tls_mandatory_ciphers and smtp_tls_mandatory_protocols parameters at this security level; all protocols are allowed and "export" grade or better ciphers are used. When TLS handshakes fail, the connection is retried with TLS disabled. This allows mail delivery to sites with non-interoperable TLS implementations.

encrypt
Mandatory TLS encryption. At this level and higher the optional "ciphers" attribute overrides the main.cf smtp_tls_mandatory_ciphers parameter and the optional "protocols" keyword overrides the main.cf smtp_tls_mandatory_protocols parameter. In the policy table, multiple protocols must be separated by colons, as attribute values may not contain whitespace or commas.

verify Mandatory TLS verification. At this security level, DNS MX lookups are trusted to be secure enough, and the name verified in the server certificate is usually obtained indirectly via unauthenticated DNS MX lookups. The optional "match" attribute overrides the main.cf smtp_tls_verify_cert_match parameter. In the policy table, multiple match patterns and strategies must be separated by colons. In practice explicit control over matching is more common with the "secure" policy, described below.

secure Secure-channel TLS. At this security level, DNS MX lookups, though potentially used to determine the candidate next-hop gateway IP addresses, are **not** trusted to be secure enough for TLS peername verification. Instead, the default name verified in the server certificate is obtained directly from the next-hop, or is explicitly specified via the optional **match** attribute which overrides the main.cf smtp_tls_secure_cert_match parameter. In the policy table, multiple match patterns and strategies must be separated by colons. The match attribute is most useful when multiple domains are supported by common server, the policy entries for additional domains specify matching rules for the primary domain certificate. While transport table overrides routing the secondary domains to the primary nexthop also allow secure verification, they risk delivery to the wrong destination when domains change hands or are re-assigned to new gateways. With the "match" attribute approach, routing is not perturbed, and mail is deferred if verification of a new MX host fails.

Example:

```
main.cf:
    smtp_tls_policy_maps = hash:/etc/postfix/tls_policy
tls_policy:
    example.edu                 none
    example.mil                 may
    example.gov                 encrypt protocols=TLSv1
    example.com                 verify ciphers=high
    example.net                 secure
    .example.net                secure match=.example.net:example.net
    [mail.example.org]:587      secure match=nexthop
```

Note: The **hostname** strategy if listed in a non-default setting of smtp_tls_secure_cert_match or in the **match** attribute in the policy table can render the **secure** level vulnerable to DNS forgery. Do not use the **hostname** strategy for secure-channel configurations in environments where DNS security is not assured.

This feature is available in Postfix 2.3 and later.

smtp_tls_scert_verifydepth (default: 5)
The verification depth for remote SMTP server certificates. A depth of 1 is sufficient, if the certificate is directly issued by a CA listed in the CA files. The default value (5) should suffice for longer chains (the root CA issues special CA which then issues the actual certificate...).

This feature is available in Postfix 2.2 and later.

smtp_tls_secure_cert_match (default: nexthop, dot-nexthop)
The server certificate peername verification method for the "secure" TLS security level. In a "secure" TLS policy table ($smtp_tls_policy_maps) entry the optional "match" attribute overrides this main.cf setting.

This parameter specifies one or more patterns or strategies separated by commas, whitespace or colons. In the policy table the only valid separator is the colon character.

For a description of the pattern and strategy syntax see the smtp_tls_verify_cert_match parameter. The "hostname" strategy should be avoided in this context, as in the absence of a secure global DNS, using the

results of MX lookups in certificate verification is not immune to active (man-in-the-middle) attacks on DNS.

Sample main.cf setting:

```
smtp_tls_secure_cert_match = nexthop
```

Sample policy table override:

```
example.net     secure match=example.com:.example.com
\&.example.net    secure match=example.com:.example.com
```

This feature is available in Postfix 2.3 and later.

smtp_tls_security_level (default: empty)

The default SMTP TLS security level for the Postfix SMTP client; when a non-empty value is specified, this overrides the obsolete parameters smtp_use_tls, smtp_enforce_tls, and smtp_tls_enforce_peername.

Specify one of the following security levels:

none TLS will not be used unless enabled for specific destinations via smtp_tls_policy_maps.

may Opportunistic TLS. TLS will be used if supported by the server. Since sending in the clear is acceptable, demanding stronger than default TLS security parameters merely reduces inter-operability. Postfix 2.3 and later ignore the smtp_tls_mandatory_ciphers and smtp_tls_mandatory_protocols parameters at this security level; all protocols are allowed and "export" grade or better ciphers are used. When TLS handshakes fail, the connection is retried with TLS disabled. This allows mail delivery to sites with non-interoperable TLS implementations.

encrypt

Mandatory TLS encryption. Since a minimum level of security is intended, it reasonable to be specific about sufficiently secure protocol versions and ciphers. At this security level and higher, the main.cf parameters smtp_tls_mandatory_protocols and smtp_tls_mandatory_ciphers specify the TLS protocols and minimum cipher grade which the administrator considers secure enough for mandatory encrypted sessions. This security level is not an appropriate default for systems delivering mail to the Internet.

verify Mandatory TLS verification. At this security level, DNS MX lookups are trusted to be secure enough, and the name verified in the server certificate is usually obtained indirectly via unauthenticated DNS MX lookups. The smtp_tls_verify_cert_match parameter controls how the server name is verified. In practice explicit control over matching is more common at the "secure" level, described below. This security level is not an appropriate default for systems delivering mail to the Internet.

secure Secure-channel TLS. At this security level, DNS MX lookups, though potentially used to determine the candidate next-hop gateway IP addresses, are **not** trusted to be secure enough for TLS peername verification. Instead, the default name verified in the server certificate is obtained from the next-hop domain as specified in the smtp_tls_secure_cert_match configuration parameter. The default matching rule is that a server certificate matches when its name is equal to or is a subdomain of the nexthop domain. This security level is not an appropriate default for systems delivering mail to the Internet.

Examples:

No TLS, old-style: smtp_use_tls=no and smtp_enforce_tls=no.

```
main.cf:
    smtp_tls_security_level = none
```

Opportunistic TLS:

```
main.cf:
    smtp_tls_security_level = may
```

Mandatory (high-grade) TLS encryption:

```
main.cf:
```

```
    smtp_tls_security_level = encrypt
    smtp_tls_mandatory_ciphers = high
```

Mandatory TLS verification, of hostname or nexthop domain:

```
main.cf:
    smtp_tls_security_level = verify
    smtp_tls_mandatory_ciphers = high
    smtp_tls_verify_cert_match = hostname, nexthop, dot-nexthop
```

Secure channel TLS with exact nexthop name matching:

```
main.cf:
    smtp_tls_security_level = secure
    smtp_tls_mandatory_protocols = TLSv1
    smtp_tls_mandatory_ciphers = high
    smtp_tls_secure_cert_match = nexthop
```

This feature is available in Postfix 2.3 and later.

smtp_tls_session_cache_database (default: empty)

Name of the file containing the optional Postfix SMTP client TLS session cache. Specify a database type that supports enumeration, such as **btree** or **sdbm**; there is no need to support concurrent access. The file is created if it does not exist. The **smtp**(8) daemon does not use this parameter directly, rather the cache is implemented indirectly in the **tlsmgr**(8) daemon. This means that per-smtp-instance master.cf overrides of this parameter are not effective. Note, that each of the cache databases supported by **tlsmgr**(8) daemon: $smtpd_tls_session_cache_database, $smtp_tls_session_cache_database (and with Postfix 2.3 and later $lmtp_session_cache_database), needs to be stored separately, it is not at this time possible to store multiple caches in a single database.

Note: **dbm** databases are not suitable. TLS session objects are too large.

Example:

```
smtp_tls_session_cache_database = btree:/var/spool/postfix/smtp_scache
```

This feature is available in Postfix 2.2 and later.

smtp_tls_session_cache_timeout (default: 3600s)

The expiration time of Postfix SMTP client TLS session cache information. A cache cleanup is performed periodically every $smtp_tls_session_cache_timeout seconds. As with $smtp_tls_session_cache_database, this parameter is implemented in the **tlsmgr**(8) daemon and therefore per-smtp-instance master.cf overrides are not possible.

This feature is available in Postfix 2.2 and later.

smtp_tls_verify_cert_match (default: hostname)

The server certificate peername verification method for the "verify" TLS security level. In a "verify" TLS policy table ($smtp_tls_policy_maps) entry the optional "match" attribute overrides this main.cf setting.

This parameter specifies one or more patterns or strategies separated by commas, whitespace or colons. In the policy table the only valid separator is the colon character.

Patterns specify domain names, or domain name suffixes:

example.com
: Match the *example.com* domain, i.e. one of the names the server certificate must be *example.com*, upper and lower case distinctions are ignored.

.example.com
: Match subdomains of the *example.com* domain, i.e. match a name in the server certificate that consists of a non-zero number of labels followed by a *.example.com* suffix. Case distinctions are ignored.

Strategies specify a transformation from the next-hop domain to the expected name in the server certificate:

nexthop
: Match against the next-hop domain, which is either the recipient domain, or the transport next-hop configured for the domain stripped of any optional socket type prefix, enclosing square brackets and trailing port. When MX lookups are not suppressed, this is the original nexthop domain prior to the MX lookup, not the result of the MX lookup. For LMTP delivery via UNIX-domain sockets, the verified next-hop name is $myhostname. This strategy is suitable for use with the "secure" policy. Case is ignored.

dot-nexthop
: As above, but match server certificate names that are subdomains of the next-hop domain. Case is ignored.

hostname
: Match against the hostname of the server, often obtained via an unauthenticated DNS MX lookup. For LMTP delivery via UNIX-domain sockets, the verified name is $myhostname. This matches the verification strategy of the "MUST" keyword in the obsolete smtp_tls_per_site table, and is suitable for use with the "verify" security level. When the next-hop name is enclosed in square brackets to suppress MX lookups, the "hostname" strategy is the same as the "nexthop" strategy. Case is ignored.

Sample main.cf setting:

```
smtp_tls_verify_cert_match = hostname, nexthop, dot-nexthop
```

Sample policy table override:

```
example.com     verify  match=hostname:nexthop
\&.example.com    verify  match=example.com:.example.com:hostname
```

This feature is available in Postfix 2.3 and later.

smtp_use_tls (default: no)

Opportunistic mode: use TLS when a remote SMTP server announces STARTTLS support, otherwise send the mail in the clear. Beware: some SMTP servers offer STARTTLS even if it is not configured. With Postfix < 2.3, if the TLS handshake fails, and no other server is available, delivery is deferred and mail stays in the queue. If this is a concern for you, use the smtp_tls_per_site feature instead.

This feature is available in Postfix 2.2 and later. With Postfix 2.3 and later use smtp_tls_security_level instead.

smtp_xforward_timeout (default: 300s)

The SMTP client time limit for sending the XFORWARD command, and for receiving the server response.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.1 and later.

smtpd_authorized_verp_clients (default: $authorized_verp_clients)

What SMTP clients are allowed to specify the XVERP command. This command requests that mail be delivered one recipient at a time with a per recipient return address.

By default, no clients are allowed to specify XVERP.

This parameter was renamed with Postfix version 2.1. The default value is backwards compatible with Postfix version 2.0.

Specify a list of network/netmask patterns, separated by commas and/or whitespace. The mask specifies the number of bits in the network part of a host address. You can also specify hostnames or \&.domain names (the initial dot causes the domain to match any name below it), "/file/name" or "type:table" patterns. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a table entry matches a lookup string (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude an address or network block from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Note: IP version 6 address information must be specified inside [] in the smtpd_authorized_verp_clients

value, and in files specified with "/file/name". IP version 6 addresses contain the ":" character, and would otherwise be confused with a "type:table" pattern.

smtpd_authorized_xclient_hosts (default: empty)

What SMTP clients are allowed to use the XCLIENT feature. This command overrides SMTP client information that is used for access control. Typical use is for SMTP-based content filters, fetchmail-like programs, or SMTP server access rule testing. See the XCLIENT_README document for details.

This feature is available in Postfix 2.1 and later.

By default, no clients are allowed to specify XCLIENT.

Specify a list of network/netmask patterns, separated by commas and/or whitespace. The mask specifies the number of bits in the network part of a host address. You can also specify hostnames or \&.domain names (the initial dot causes the domain to match any name below it), "/file/name" or "type:table" patterns. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a table entry matches a lookup string (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude an address or network block from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Note: IP version 6 address information must be specified inside [] in the smtpd_authorized_xclient_hosts value, and in files specified with "/file/name". IP version 6 addresses contain the ":" character, and would otherwise be confused with a "type:table" pattern.

smtpd_authorized_xforward_hosts (default: empty)

What SMTP clients are allowed to use the XFORWARD feature. This command forwards information that is used to improve logging after SMTP-based content filters. See the XFORWARD_README document for details.

This feature is available in Postfix 2.1 and later.

By default, no clients are allowed to specify XFORWARD.

Specify a list of network/netmask patterns, separated by commas and/or whitespace. The mask specifies the number of bits in the network part of a host address. You can also specify hostnames or \&.domain names (the initial dot causes the domain to match any name below it), "/file/name" or "type:table" patterns. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a table entry matches a lookup string (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude an address or network block from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Note: IP version 6 address information must be specified inside [] in the smtpd_authorized_xforward_hosts value, and in files specified with "/file/name". IP version 6 addresses contain the ":" character, and would otherwise be confused with a "type:table" pattern.

smtpd_banner (default: $myhostname ESMTP $mail_name)

The text that follows the 220 status code in the SMTP greeting banner. Some people like to see the mail version advertised. By default, Postfix shows no version.

You MUST specify $myhostname at the start of the text. This is required by the SMTP protocol.

Example:

```
smtpd_banner = $myhostname ESMTP $mail_name ($mail_version)
```

smtpd_client_connection_count_limit (default: 50)

How many simultaneous connections any client is allowed to make to this service. By default, the limit is set to half the default process limit value.

To disable this feature, specify a limit of 0.

WARNING: The purpose of this feature is to limit abuse. It must not be used to regulate legitimate mail traffic.

This feature is available in Postfix 2.2 and later.

smtpd_client_connection_rate_limit (default: 0)

The maximal number of connection attempts any client is allowed to make to this service per time unit. The time unit is specified with the anvil_rate_time_unit configuration parameter.

By default, a client can make as many connections per time unit as Postfix can accept.

To disable this feature, specify a limit of 0.

WARNING: The purpose of this feature is to limit abuse. It must not be used to regulate legitimate mail traffic.

This feature is available in Postfix 2.2 and later.

Example:

```
smtpd_client_connection_rate_limit = 1000
```

smtpd_client_event_limit_exceptions (default: $mynetworks)

Clients that are excluded from connection count, connection rate, or SMTP request rate restrictions. See the mynetworks parameter description for the parameter value syntax.

By default, clients in trusted networks are excluded. Specify a list of network blocks, hostnames or .domain names (the initial dot causes the domain to match any name below it).

Note: IP version 6 address information must be specified inside [] in the smtpd_client_event_limit_exceptions value, and in files specified with "/file/name". IP version 6 addresses contain the ":" character, and would otherwise be confused with a "type:table" pattern.

This feature is available in Postfix 2.2 and later.

smtpd_client_message_rate_limit (default: 0)

The maximal number of message delivery requests that any client is allowed to make to this service per time unit, regardless of whether or not Postfix actually accepts those messages. The time unit is specified with the anvil_rate_time_unit configuration parameter.

By default, a client can send as many message delivery requests per time unit as Postfix can accept.

To disable this feature, specify a limit of 0.

WARNING: The purpose of this feature is to limit abuse. It must not be used to regulate legitimate mail traffic.

This feature is available in Postfix 2.2 and later.

Example:

```
smtpd_client_message_rate_limit = 1000
```

smtpd_client_new_tls_session_rate_limit (default: 0)

The maximal number of new (i.e., uncached) TLS sessions that a remote SMTP client is allowed to negotiate with this service per time unit. The time unit is specified with the anvil_rate_time_unit configuration parameter.

By default, a remote SMTP client can negotiate as many new TLS sessions per time unit as Postfix can accept.

To disable this feature, specify a limit of 0. Otherwise, specify a limit that is at least the per-client concurrent session limit, or else legitimate client sessions may be rejected.

WARNING: The purpose of this feature is to limit abuse. It must not be used to regulate legitimate mail traffic.

This feature is available in Postfix 2.3 and later.

Example:

```
smtpd_client_new_tls_session_rate_limit = 100
```

smtpd_client_recipient_rate_limit (default: 0)

The maximal number of recipient addresses that any client is allowed to send to this service per time unit, regardless of whether or not Postfix actually accepts those recipients. The time unit is specified with the anvil_rate_time_unit configuration parameter.

By default, a client can make as many recipient addresses per time unit as Postfix can accept.

To disable this feature, specify a limit of 0.

WARNING: The purpose of this feature is to limit abuse. It must not be used to regulate legitimate mail traffic.

This feature is available in Postfix 2.2 and later.

Example:

```
smtpd_client_recipient_rate_limit = 1000
```

smtpd_client_restrictions (default: empty)

Optional SMTP server access restrictions in the context of a client SMTP connection request.

The default is to allow all connection requests.

Specify a list of restrictions, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. Restrictions are applied in the order as specified; the first restriction that matches wins.

The following restrictions are specific to client hostname or client network address information.

check_ccert_access *type:table*

Use the client certificate fingerprint as lookup key for the specified **access**(5) database; with Postfix version 2.2, also require that the SMTP client certificate is verified successfully. This feature is available with Postfix version 2.2 and later.

check_client_access *type:table*

Search the specified access database for the client hostname, parent domains, client IP address, or networks obtained by stripping least significant octets. See the **access**(5) manual page for details.

permit_inet_interfaces

Permit the request when the client IP address matches $inet_interfaces.

permit_mynetworks

Permit the request when the client IP address matches any network or network address listed in $mynetworks.

permit_sasl_authenticated

Permit the request when the client is successfully authenticated via the RFC 2554 (AUTH) protocol.

permit_tls_all_clientcerts

Permit the request when the remote SMTP client certificate is verified successfully. This option must be used only if a special CA issues the certificates and only this CA is listed as trusted CA, otherwise all clients with a recognized certificate would be allowed to relay. This feature is available with Postfix version 2.2.

permit_tls_clientcerts

Permit the request when the remote SMTP client certificate is verified successfully, and the certificate fingerprint is listed in $relay_clientcerts. This feature is available with Postfix version 2.2.

reject_rbl_client *rbl_domain=d.d.d.d*

Reject the request when the reversed client network address is listed with the A record "*d.d.d.d*" under *rbl_domain* (Postfix version 2.1 and later only). If no "*=d.d.d.d*" is specified, reject the request when the reversed client network address is listed with any A record under *rbl_domain*.
The maps_rbl_reject_code parameter specifies the response code for rejected requests (default: 554), the default_rbl_reply parameter specifies the default server reply, and the rbl_reply_maps

parameter specifies tables with server replies indexed by *rbl_domain*. This feature is available in Postfix 2.0 and later.

reject_rhsbl_client *rbl_domain=d.d.d.d*
Reject the request when the client hostname is listed with the A record "*d.d.d.d*" under *rbl_domain* (Postfix version 2.1 and later only). If no "*=d.d.d.d*" is specified, reject the request when the client hostname is listed with any A record under *rbl_domain*. See the reject_rbl_client description above for additional RBL related configuration parameters. This feature is available in Postfix 2.0 and later.

reject_unknown_client_hostname (with Postfix < 2.3: reject_unknown_client)
Reject the request when 1) the client IP address->name mapping fails, 2) the name->address mapping fails, or 3) the name->address mapping does not match the client IP address.
This is a stronger restriction than the reject_unknown_reverse_client_hostname feature, which triggers only under condition 1) above.
The unknown_client_reject_code parameter specifies the response code for rejected requests (default: 450). The reply is always 450 in case the address->name or name->address lookup failed due to a temporary problem.

reject_unknown_reverse_client_hostname
Reject the request when the client IP address has no address->name mapping.
This is a weaker restriction than the reject_unknown_client_hostname feature, which requires not only that the address->name and name->address mappings exist, but also that the two mappings reproduce the client IP address.
The unknown_client_reject_code parameter specifies the response code for rejected requests (default: 450). The reply is always 450 in case the address->name lookup failed due to a temporary problem.
This feature is available in Postfix 2.3 and later.

In addition, you can use any of the following generic restrictions. These restrictions are applicable in any SMTP command context.

check_policy_service *servername*
Query the specified policy server. See the SMTPD_POLICY_README document for details. This feature is available in Postfix 2.1 and later.

defer Defer the request. The client is told to try again later. This restriction is useful at the end of a restriction list, to make the default policy explicit.
The defer_code parameter specifies the SMTP server reply code (default: 450).

defer_if_permit
Defer the request if some later restriction would result in an explicit or implicit PERMIT action. This is useful when a blacklisting feature fails due to a temporary problem. This feature is available in Postfix version 2.1 and later.

defer_if_reject
Defer the request if some later restriction would result in a REJECT action. This is useful when a whitelisting feature fails due to a temporary problem. This feature is available in Postfix version 2.1 and later.

permit Permit the request. This restriction is useful at the end of a restriction list, to make the default policy explicit.

reject_multi_recipient_bounce
Reject the request when the envelope sender is the null address, and the message has multiple envelope recipients. This usage has rare but legitimate applications: under certain conditions, multi-recipient mail that was posted with the DSN option NOTIFY=NEVER may be forwarded with the null sender address.
Note: this restriction can only work reliably when used in smtpd_data_restrictions or smtpd_end_of_data_restrictions, because the total number of recipients is not known at an earlier

stage of the SMTP conversation. Use at the RCPT stage will only reject the second etc. recipient. The multi_recipient_bounce_reject_code parameter specifies the response code for rejected requests (default: 550). This feature is available in Postfix 2.1 and later.

reject_plaintext_session
Reject the request when the connection is not encrypted. This restriction should not be used before the client has had a chance to negotiate encryption with the AUTH or STARTTLS commands.
The plaintext_reject_code parameter specifies the response code for rejected requests (default: 450). This feature is available in Postfix 2.3 and later.

reject_unauth_pipelining
Reject the request when the client sends SMTP commands ahead of time where it is not allowed, or when the client sends SMTP commands ahead of time without knowing that Postfix actually supports ESMTP command pipelining. This stops mail from bulk mail software that improperly uses ESMTP command pipelining in order to speed up deliveries.
Note: reject_unauth_pipelining is not useful outside smtpd_data_restrictions when 1) the client uses ESMTP (EHLO instead of HELO) and 2) with "smtpd_delay_reject = yes" (the default). The use of reject_unauth_pipelining in the other restriction contexts is therefore not recommended.

reject Reject the request. This restriction is useful at the end of a restriction list, to make the default policy explicit. The reject_code configuration parameter specifies the response code to rejected requests (default: 554).

sleep *seconds*
Pause for the specified number of seconds and proceed with the next restriction in the list, if any. This may stop zombie mail when used as:

```
/etc/postfix/main.cf:
    smtpd_client_restrictions =
        sleep 1, reject_unauth_pipelining
    smtpd_delay_reject = no
```

This feature is available in Postfix 2.3.

warn_if_reject
Change the meaning of the next restriction, so that it logs a warning instead of rejecting a request (look for logfile records that contain "reject_warning"). This is useful for testing new restrictions in a "live" environment without risking unnecessary loss of mail.

Other restrictions that are valid in this context:

- SMTP command specific restrictions that are described under the smtpd_helo_restrictions, smtpd_sender_restrictions or smtpd_recipient_restrictions parameters. When helo, sender or recipient restrictions are listed under smtpd_client_restrictions, they have effect only with "smtpd_delay_reject = yes", so that $smtpd_client_restrictions is evaluated at the time of the RCPT TO command.

Example:

```
smtpd_client_restrictions = permit_mynetworks, reject_unknown_client_hostname
```

smtpd_data_restrictions (default: empty)
Optional access restrictions that the Postfix SMTP server applies in the context of the SMTP DATA command.

This feature is available in Postfix 2.0 and later.

Specify a list of restrictions, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. Restrictions are applied in the order as specified; the first restriction that matches wins.

The following restrictions are valid in this context:

- Generic restrictions that can be used in any SMTP command context, described under smtpd_client_restrictions.
- SMTP command specific restrictions described under smtpd_client_restrictions, smtpd_helo_restrictions, smtpd_sender_restrictions or smtpd_recipient_restrictions.

Examples:

```
smtpd_data_restrictions = reject_unauth_pipelining
smtpd_data_restrictions = reject_multi_recipient_bounce
```

smtpd_delay_open_until_valid_rcpt (default: yes)

Postpone the start of an SMTP mail transaction until a valid RCPT TO command is received. Specify "no" to create a mail transaction as soon as the SMTP server receives a valid MAIL FROM command.

With sites that reject lots of mail, the default setting reduces the use of disk, CPU and memory resources. The downside is that rejected recipients are logged with NOQUEUE instead of a mail transaction ID. This complicates the logfile analysis of multi-recipient mail.

This feature is available in Postfix 2.3 and later.

smtpd_delay_reject (default: yes)

Wait until the RCPT TO command before evaluating $smtpd_client_restrictions, $smtpd_helo_restrictions and $smtpd_sender_restrictions, or wait until the ETRN command before evaluating $smtpd_client_restrictions and $smtpd_helo_restrictions.

This feature is turned on by default because some clients apparently mis-behave when the Postfix SMTP server rejects commands before RCPT TO.

The default setting has one major benefit: it allows Postfix to log recipient address information when rejecting a client name/address or sender address, so that it is possible to find out whose mail is being rejected.

smtpd_discard_ehlo_keyword_address_maps (default: empty)

Lookup tables, indexed by the remote SMTP client address, with case insensitive lists of EHLO keywords (pipelining, starttls, auth, etc.) that the SMTP server will not send in the EHLO response to a remote SMTP client. See smtpd_discard_ehlo_keywords for details. The table is not searched by hostname for robustness reasons.

This feature is available in Postfix 2.2 and later.

smtpd_discard_ehlo_keywords (default: empty)

A case insensitive list of EHLO keywords (pipelining, starttls, auth, etc.) that the SMTP server will not send in the EHLO response to a remote SMTP client.

This feature is available in Postfix 2.2 and later.

Notes:

- Specify the **silent-discard** pseudo keyword to prevent this action from being logged.
- Use the smtpd_discard_ehlo_keyword_address_maps feature to discard EHLO keywords selectively.

smtpd_end_of_data_restrictions (default: empty)

Optional access restrictions that the Postfix SMTP server applies in the context of the SMTP END-OF-DATA command.

This feature is available in Postfix 2.2 and later.

See smtpd_data_restrictions for syntax details.

smtpd_enforce_tls (default: no)

Mandatory TLS: announce STARTTLS support to SMTP clients, and require that clients use TLS encryption. According to RFC 2487 this MUST NOT be applied in case of a publicly-referenced SMTP server. This option is off by default and should be used only on dedicated servers.

Note 1: "smtpd_enforce_tls = yes" implies "smtpd_tls_auth_only = yes".

Note 2: when invoked via "**sendmail -bs**", Postfix will never offer STARTTLS due to insufficient privileges to access the server private key. This is intended behavior.

This feature is available in Postfix 2.2 and later. With Postfix 2.3 and later use smtpd_tls_security_level instead.

smtpd_error_sleep_time (default: 1s)

With Postfix version 2.1 and later: the SMTP server response delay after a client has made more than $smtpd_soft_error_limit errors, and fewer than $smtpd_hard_error_limit errors, without delivering mail.

With Postfix version 2.0 and earlier: the SMTP server delay before sending a reject (4xx or 5xx) response, when the client has made fewer than $smtpd_soft_error_limit errors without delivering mail.

smtpd_etrn_restrictions (default: empty)

Optional SMTP server access restrictions in the context of a client ETRN request.

The Postfix ETRN implementation accepts only destinations that are eligible for the Postfix "fast flush" service. See the ETRN_README file for details.

Specify a list of restrictions, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. Restrictions are applied in the order as specified; the first restriction that matches wins.

The following restrictions are specific to the domain name information received with the ETRN command.

check_etrn_access *type:table*

Search the specified access database for the ETRN domain name or its parent domains. See the **access**(5) manual page for details.

Other restrictions that are valid in this context:

- Generic restrictions that can be used in any SMTP command context, described under smtpd_client_restrictions.
- SMTP command specific restrictions described under smtpd_client_restrictions and smtpd_helo_restrictions.

Example:

```
smtpd_etrn_restrictions = permit_mynetworks, reject
```

smtpd_expansion_filter (default: see postconf -d output)

What characters are allowed in $name expansions of RBL reply templates. Characters not in the allowed set are replaced by "_". Use C like escapes to specify special characters such as whitespace.

This parameter is not subjected to $parameter expansion.

This feature is available in Postfix 2.0 and later.

smtpd_forbidden_commands (default: CONNECT, GET, POST)

List of commands that causes the Postfix SMTP server to immediately terminate the session with a 221 code. This can be used to disconnect clients that obviously attempt to abuse the system. In addition to the commands listed in this parameter, commands that follow the "Label:" format of message headers will also cause a disconnect.

This feature is available in Postfix 2.2 and later.

smtpd_hard_error_limit (default: 20)

The maximal number of errors a remote SMTP client is allowed to make without delivering mail. The Postfix SMTP server disconnects when the limit is exceeded.

smtpd_helo_required (default: no)

Require that a remote SMTP client introduces itself at the beginning of an SMTP session with the HELO or EHLO command.

Example:

```
smtpd_helo_required = yes
```

smtpd_helo_restrictions (default: empty)

Optional restrictions that the Postfix SMTP server applies in the context of the SMTP HELO command.

The default is to permit everything.

Specify a list of restrictions, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. Restrictions are applied in the order as specified; the first restriction that matches wins.

The following restrictions are specific to the hostname information received with the HELO or EHLO command.

check_helo_access *type:table*

Search the specified **access**(5) database for the HELO or EHLO hostname or parent domains, and execute the corresponding action.

check_helo_mx_access *type:table*

Search the specified **access**(5) database for the MX hosts for the HELO or EHLO hostname, and execute the corresponding action. Note: a result of "OK" is not allowed for safety reasons. Instead, use DUNNO in order to exclude specific hosts from blacklists. This feature is available in Postfix 2.1 and later.

check_helo_ns_access *type:table*

Search the specified **access**(5) database for the DNS servers for the HELO or EHLO hostname, and execute the corresponding action. Note: a result of "OK" is not allowed for safety reasons. Instead, use DUNNO in order to exclude specific hosts from blacklists. This feature is available in Postfix 2.1 and later.

reject_invalid_helo_hostname (with Postfix < 2.3: reject_invalid_hostname)

Reject the request when the HELO or EHLO hostname syntax is invalid.
The invalid_hostname_reject_code specifies the response code to rejected requests (default: 501).

reject_non_fqdn_helo_hostname (with Postfix < 2.3: reject_non_fqdn_hostname)

Reject the request when the HELO or EHLO hostname is not in fully-qualified domain form, as required by the RFC.
The non_fqdn_reject_code parameter specifies the response code to rejected requests (default: 504).

reject_unknown_helo_hostname (with Postfix < 2.3: reject_unknown_hostname)

Reject the request when the HELO or EHLO hostname has no DNS A or MX record.
The unknown_hostname_reject_code specifies the response code to rejected requests (default: 450).

Other restrictions that are valid in this context:

- Generic restrictions that can be used in any SMTP command context, described under smtpd_client_restrictions.
- Client hostname or network address specific restrictions described under smtpd_client_restrictions.
- SMTP command specific restrictions described under smtpd_sender_restrictions or smtpd_recipient_restrictions. When sender or recipient restrictions are listed under smtpd_helo_restrictions, they have effect only with "smtpd_delay_reject = yes", so that $smtpd_helo_restrictions is evaluated at the time of the RCPT TO command.

Examples:

```
smtpd_helo_restrictions = permit_mynetworks, reject_invalid_helo_hostname
smtpd_helo_restrictions = permit_mynetworks, reject_unknown_helo_hostname
```

smtpd_history_flush_threshold (default: 100)

The maximal number of lines in the Postfix SMTP server command history before it is flushed upon receipt of EHLO, RSET, or end of DATA.

smtpd_junk_command_limit (default: 100)
The number of junk commands (NOOP, VRFY, ETRN or RSET) that a remote SMTP client can send before the Postfix SMTP server starts to increment the error counter with each junk command. The junk command count is reset after mail is delivered. See also the smtpd_error_sleep_time and smtpd_soft_error_limit configuration parameters.

smtpd_milters (default: empty)
A list of Milter (mail filter) applications for new mail that arrives via the Postfix **smtpd**(8) server. See the MILTER_README document for details.

This feature is available in Postfix 2.3 and later.

smtpd_noop_commands (default: empty)
List of commands that the Postfix SMTP server replies to with "250 Ok", without doing any syntax checks and without changing state. This list overrides any commands built into the Postfix SMTP server.

smtpd_null_access_lookup_key (default: <>)
The lookup key to be used in SMTP **access**(5) tables instead of the null sender address.

smtpd_peername_lookup (default: yes)
Attempt to look up the remote SMTP client hostname, and verify that the name matches the client IP address. A client name is set to "unknown" when it cannot be looked up or verified, or when name lookup is disabled. Turning off name lookup reduces delays due to DNS lookup and increases the maximal inbound delivery rate.

This feature is available in Postfix 2.3 and later.

smtpd_policy_service_max_idle (default: 300s)
The time after which an idle SMTPD policy service connection is closed.

This feature is available in Postfix 2.1 and later.

smtpd_policy_service_max_ttl (default: 1000s)
The time after which an active SMTPD policy service connection is closed.

This feature is available in Postfix 2.1 and later.

smtpd_policy_service_timeout (default: 100s)
The time limit for connecting to, writing to or receiving from a delegated SMTPD policy server.

This feature is available in Postfix 2.1 and later.

smtpd_proxy_ehlo (default: $myhostname)
How the Postfix SMTP server announces itself to the proxy filter. By default, the Postfix hostname is used.

This feature is available in Postfix 2.1 and later.

smtpd_proxy_filter (default: empty)
The hostname and TCP port of the mail filtering proxy server. The proxy receives all mail from the Postfix SMTP server, and is supposed to give the result to another Postfix SMTP server process.

Specify "host:port" or "inet:host:port" for a TCP endpoint, or "unix:pathname" for a UNIX-domain endpoint. The host can be specified as an IP address or as a symbolic name; no MX lookups are done. When no "host" or "host:" are specified, the local machine is assumed. Pathname interpretation is relative to the Postfix queue directory.

This feature is available in Postfix 2.1 and later.

The "inet:" and "unix:" prefixes are available in Postfix 2.3 and later.

smtpd_proxy_timeout (default: 100s)
The time limit for connecting to a proxy filter and for sending or receiving information. When a connection fails the client gets a generic error message while more detailed information is logged to the maillog file.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

This feature is available in Postfix 2.1 and later.

smtpd_recipient_limit (default: 1000)
The maximal number of recipients that the Postfix SMTP server accepts per message delivery request.

smtpd_recipient_overshoot_limit (default: 1000)
The number of recipients that a remote SMTP client can send in excess of the limit specified with $smtpd_recipient_limit, before the Postfix SMTP server increments the per-session error count for each excess recipient.

smtpd_recipient_restrictions (default: permit_mynetworks, reject_unauth_destination)
The access restrictions that the Postfix SMTP server applies in the context of the RCPT TO command.

By default, the Postfix SMTP server accepts:

- Mail from clients whose IP address matches $mynetworks, or:
- Mail to remote destinations that match $relay_domains, except for addresses that contain sender-specified routing (user@elsewhere@domain), or:
- Mail to local destinations that match $inet_interfaces or $proxy_interfaces, $mydestination, $virtual_alias_domains, or $virtual_mailbox_domains.

IMPORTANT: If you change this parameter setting, you must specify at least one of the following restrictions. Otherwise Postfix will refuse to receive mail:

```
reject, defer, defer_if_permit, reject_unauth_destination
```

Specify a list of restrictions, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. Restrictions are applied in the order as specified; the first restriction that matches wins.

The following restrictions are specific to the recipient address that is received with the RCPT TO command.

check_recipient_access *type:table*
Search the specified **access**(5) database for the resolved RCPT TO address, domain, parent domains, or localpart@, and execute the corresponding action.

check_recipient_mx_access *type:table*
Search the specified **access**(5) database for the MX hosts for the RCPT TO address, and execute the corresponding action. Note: a result of "OK" is not allowed for safety reasons. Instead, use DUNNO in order to exclude specific hosts from blacklists. This feature is available in Postfix 2.1 and later.

check_recipient_ns_access *type:table*
Search the specified **access**(5) database for the DNS servers for the RCPT TO address, and execute the corresponding action. Note: a result of "OK" is not allowed for safety reasons. Instead, use DUNNO in order to exclude specific hosts from blacklists. This feature is available in Postfix 2.1 and later.

permit_auth_destination
Permit the request when one of the following is true:

- Postfix is mail forwarder: the resolved RCPT TO address matches $relay_domains or a subdomain thereof, and the address contains no sender-specified routing (user@elsewhere@domain),
- Postfix is the final destination: the resolved RCPT TO address matches $mydestination, $inet_interfaces, $proxy_interfaces, $virtual_alias_domains, or $virtual_mailbox_domains, and the address contains no sender-specified routing (user@elsewhere@domain).

permit_mx_backup
Permit the request when the local mail system is backup MX for the RCPT TO address, or when the address is an authorized destination (see permit_auth_destination for definition).

- Safety: permit_mx_backup does not accept addresses that have sender-specified routing information (example: user@elsewhere@domain).

- Safety: permit_mx_backup can be vulnerable to mis-use when access is not restricted with permit_mx_backup_networks.
- Safety: as of Postfix version 2.3, permit_mx_backup no longer accepts the address when the local mail system is primary MX for the recipient domain. Exception: permit_mx_backup accepts the address when it specifies an authorized destination (see permit_auth_destination for definition).
- Limitation: mail may be rejected in case of a temporary DNS lookup problem with Postfix prior to version 2.0.

reject_non_fqdn_recipient
Reject the request when the RCPT TO address is not in fully-qualified domain form, as required by the RFC.
The non_fqdn_reject_code parameter specifies the response code to rejected requests (default: 504).

reject_rhsbl_recipient *rbl_domain=d.d.d.d*
Reject the request when the RCPT TO domain is listed with the A record "*d.d.d.d*" under *rbl_domain* (Postfix version 2.1 and later only). If no "*=d.d.d.d*" is specified, reject the request when the RCPT TO domain is listed with any A record under *rbl_domain*.
The maps_rbl_reject_code parameter specifies the response code for rejected requests (default: 554); the default_rbl_reply parameter specifies the default server reply; and the rbl_reply_maps parameter specifies tables with server replies indexed by *rbl_domain*. This feature is available in Postfix version 2.0 and later.

reject_unauth_destination
Reject the request unless one of the following is true:

- Postfix is mail forwarder: the resolved RCPT TO address matches $relay_domains or a subdomain thereof, and contains no sender-specified routing (user@elsewhere@domain),
- Postfix is the final destination: the resolved RCPT TO address matches $mydestination, $inet_interfaces, $proxy_interfaces, $virtual_alias_domains, or $virtual_mailbox_domains, and contains no sender-specified routing (user@elsewhere@domain).
  The relay_domains_reject_code parameter specifies the response code for rejected requests (default: 554).

reject_unknown_recipient_domain
Reject the request when Postfix is not final destination for the recipient address, and the RCPT TO address has no DNS A or MX record, or when it has a malformed MX record such as a record with a zero-length MX hostname (Postfix version 2.3 and later).
The unknown_address_reject_code parameter specifies the response code for rejected requests (default: 450). The response is always 450 in case of a temporary DNS error.

reject_unlisted_recipient (with Postfix version 2.0: check_recipient_maps)
Reject the request when the RCPT TO address is not listed in the list of valid recipients for its domain class. See the smtpd_reject_unlisted_recipient parameter description for details. This feature is available in Postfix 2.1 and later.

reject_unverified_recipient
Reject the request when mail to the RCPT TO address is known to bounce, or when the recipient address destination is not reachable. Address verification information is managed by the **verify**(8) server; see the ADDRESS_VERIFICATION_README file for details.
The unverified_recipient_reject_code parameter specifies the response when an address is known to bounce (default: 450, change into 550 when you are confident that it is safe to do so). Postfix replies with 450 when an address probe failed due to a temporary problem. This feature is available in Postfix 2.1 and later.

Other restrictions that are valid in this context:

- Generic restrictions that can be used in any SMTP command context, described under smtpd_client_restrictions.

- SMTP command specific restrictions described under smtpd_client_restrictions, smtpd_helo_restrictions and smtpd_sender_restrictions.

Example:

```
smtpd_recipient_restrictions = permit_mynetworks, reject_unauth_destination
```

smtpd_reject_unlisted_recipient (default: yes)

Request that the Postfix SMTP server rejects mail for unknown recipient addresses, even when no explicit reject_unlisted_recipient access restriction is specified. This prevents the Postfix queue from filling up with undeliverable MAILER-DAEMON messages.

- The recipient domain matches $mydestination, $inet_interfaces or $proxy_interfaces, but the recipient is not listed in $local_recipient_maps, and $local_recipient_maps is not null.
- The recipient domain matches $virtual_alias_domains but the recipient is not listed in $virtual_alias_maps.
- The recipient domain matches $virtual_mailbox_domains but the recipient is not listed in $virtual_mailbox_maps, and $virtual_mailbox_maps is not null.
- The recipient domain matches $relay_domains but the recipient is not listed in $relay_recipient_maps, and $relay_recipient_maps is not null.

This feature is available in Postfix 2.1 and later.

smtpd_reject_unlisted_sender (default: no)

Request that the Postfix SMTP server rejects mail from unknown sender addresses, even when no explicit reject_unlisted_sender access restriction is specified. This can slow down an explosion of forged mail from worms or viruses.

- The sender domain matches $mydestination, $inet_interfaces or $proxy_interfaces, but the sender is not listed in $local_recipient_maps, and $local_recipient_maps is not null.
- The sender domain matches $virtual_alias_domains but the sender is not listed in $virtual_alias_maps.
- The sender domain matches $virtual_mailbox_domains but the sender is not listed in $virtual_mailbox_maps, and $virtual_mailbox_maps is not null.
- The sender domain matches $relay_domains but the sender is not listed in $relay_recipient_maps, and $relay_recipient_maps is not null.

This feature is available in Postfix 2.1 and later.

smtpd_restriction_classes (default: empty)

User-defined aliases for groups of access restrictions. The aliases can be specified in smtpd_recipient_restrictions etc., and on the right-hand side of a Postfix **access**(5) table.

One major application is for implementing per-recipient UCE control. See the RESTRICTION_CLASS_README document for other examples.

smtpd_sasl_application_name (default: smtpd)

The application name used for SASL server initialization. This controls the name of the SASL configuration file. The default value is **smtpd**, corresponding to a SASL configuration file named **smtpd.conf**.

This feature is available in Postfix 2.1 and 2.2. With Postfix 2.3 it was renamed to smtpd_sasl_path.

smtpd_sasl_auth_enable (default: no)

Enable SASL authentication in the Postfix SMTP server. By default, the Postfix SMTP server does not use authentication.

If a remote SMTP client is authenticated, the permit_sasl_authenticated access restriction can be used to permit relay access, like this:

```
smtpd_recipient_restrictions =
    permit_mynetworks, permit_sasl_authenticated, ...
```

To reject all SMTP connections from unauthenticated clients, specify "smtpd_delay_reject = yes" (which is the default) and use:

```
smtpd_client_restrictions = permit_sasl_authenticated, reject
```

See the SASL_README file for SASL configuration and operation details.

smtpd_sasl_authenticated_header (default: no)

Report the SASL authenticated user name in the **smtpd**(8) Received message header.

This feature is available in Postfix 2.3 and later.

smtpd_sasl_exceptions_networks (default: empty)

What SMTP clients Postfix will not offer AUTH support to.

Some clients (Netscape 4 at least) have a bug that causes them to require a login and password whenever AUTH is offered, whether it's necessary or not. To work around this, specify, for example, $mynetworks to prevent Postfix from offering AUTH to local clients.

Specify a list of network/netmask patterns, separated by commas and/or whitespace. The mask specifies the number of bits in the network part of a host address. You can also "/file/name" or "type:table" patterns. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a table entry matches a lookup string (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude an address or network block from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

Note: IP version 6 address information must be specified inside [] in the smtpd_sasl_exceptions_networks value, and in files specified with "/file/name". IP version 6 addresses contain the ":" character, and would otherwise be confused with a "type:table" pattern.

Example:

```
smtpd_sasl_exceptions_networks = $mynetworks
```

This feature is available in Postfix 2.1 and later.

smtpd_sasl_local_domain (default: empty)

The name of the local SASL authentication realm.

By default, the local authentication realm name is the null string.

Examples:

```
smtpd_sasl_local_domain = $mydomain
smtpd_sasl_local_domain = $myhostname
```

smtpd_sasl_path (default: smtpd)

Implementation-specific information that is passed through to the SASL plug-in implementation that is selected with **smtpd_sasl_type**. Typically this specifies the name of a configuration file or rendezvous point.

This feature is available in Postfix 2.3 and later. In earlier releases it was called smtpd_sasl_application.

smtpd_sasl_security_options (default: noanonymous)

SASL security options; as of Postfix 2.3 the list of available features depends on the SASL server implementation that is selected with **smtpd_sasl_type**.

The following security features are defined for the **cyrus** server SASL implementation:

Restrict what authentication mechanisms the Postfix SMTP server will offer to the client. The list of available authentication mechanisms is system dependent.

Specify zero or more of the following:

noplaintext

Disallow methods that use plaintext passwords.

noactive
Disallow methods subject to active (non-dictionary) attack.

nodictionary
Disallow methods subject to passive (dictionary) attack.

noanonymous
Disallow methods that allow anonymous authentication.

mutual_auth
Only allow methods that provide mutual authentication (not available with SASL version 1).

By default, the Postfix SMTP server accepts plaintext passwords but not anonymous logins.

Warning: it appears that clients try authentication methods in the order as advertised by the server (e.g., PLAIN ANONYMOUS CRAM-MD5) which means that if you disable plaintext passwords, clients will log in anonymously, even when they should be able to use CRAM-MD5. So, if you disable plaintext logins, disable anonymous logins too. Postfix treats anonymous login as no authentication.

Example:

```
smtpd_sasl_security_options = noanonymous, noplaintext
```

smtpd_sasl_tls_security_options (default: $smtpd_sasl_security_options)
The SASL authentication security options that the Postfix SMTP server uses for TLS encrypted SMTP sessions.

This feature is available in Postfix 2.2 and later.

smtpd_sasl_type (default: cyrus)
The SASL plug-in type that the Postfix SMTP server should use for authentication. The available types are listed with the "**postconf -a**" command.

This feature is available in Postfix 2.3 and later.

smtpd_sender_login_maps (default: empty)
Optional lookup table with the SASL login names that own sender (MAIL FROM) addresses.

Specify zero or more "type:table" lookup tables. With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, the following search operations are done with a sender address of *user@domain*:

1) *user@domain*
This table lookup is always done and has the highest precedence.

2) *user* This table lookup is done only when the *domain* part of the sender address matches $myorigin, $mydestination, $inet_interfaces or $proxy_interfaces.

3) *@domain*
This table lookup is done last and has the lowest precedence.

In all cases the result of table lookup must be either "not found" or a list of SASL login names separated by comma and/or whitespace.

smtpd_sender_restrictions (default: empty)
Optional restrictions that the Postfix SMTP server applies in the context of the MAIL FROM command.

The default is to permit everything.

Specify a list of restrictions, separated by commas and/or whitespace. Continue long lines by starting the next line with whitespace. Restrictions are applied in the order as specified; the first restriction that matches wins.

The following restrictions are specific to the sender address received with the MAIL FROM command.

check_sender_access *type:table*
Search the specified **access**(5) database for the MAIL FROM address, domain, parent domains, or localpart@, and execute the corresponding action.

check_sender_mx_access *type:table*
Search the specified **access**(5) database for the MX hosts for the MAIL FROM address, and execute the corresponding action. Note: a result of "OK" is not allowed for safety reasons. Instead, use DUNNO in order to exclude specific hosts from blacklists. This feature is available in Postfix 2.1 and later.

check_sender_ns_access *type:table*
Search the specified **access**(5) database for the DNS servers for the MAIL FROM address, and execute the corresponding action. Note: a result of "OK" is not allowed for safety reasons. Instead, use DUNNO in order to exclude specific hosts from blacklists. This feature is available in Postfix 2.1 and later.

reject_authenticated_sender_login_mismatch
Enforces the reject_sender_login_mismatch restriction for authenticated clients only. This feature is available in Postfix version 2.1 and later.

reject_non_fqdn_sender
Reject the request when the MAIL FROM address is not in fully-qualified domain form, as required by the RFC.
The non_fqdn_reject_code parameter specifies the response code to rejected requests (default: 504).

reject_rhsbl_sender *rbl_domain=d.d.d.d*
Reject the request when the MAIL FROM domain is listed with the A record "*d.d.d.d*" under *rbl_domain* (Postfix version 2.1 and later only). If no "*=d.d.d.d*" is specified, reject the request when the MAIL FROM domain is listed with any A record under *rbl_domain*.
The maps_rbl_reject_code parameter specifies the response code for rejected requests (default: 554); the default_rbl_reply parameter specifies the default server reply; and the rbl_reply_maps parameter specifies tables with server replies indexed by *rbl_domain*. This feature is available in Postfix 2.0 and later.

reject_sender_login_mismatch
Reject the request when $smtpd_sender_login_maps specifies an owner for the MAIL FROM address, but the client is not (SASL) logged in as that MAIL FROM address owner; or when the client is (SASL) logged in, but the client login name doesn’t own the MAIL FROM address according to $smtpd_sender_login_maps.

reject_unauthenticated_sender_login_mismatch
Enforces the reject_sender_login_mismatch restriction for unauthenticated clients only. This feature is available in Postfix version 2.1 and later.

reject_unknown_sender_domain
Reject the request when Postfix is not final destination for the sender address, and the MAIL FROM address has no DNS A or MX record, or when it has a malformed MX record such as a record with a zero-length MX hostname (Postfix version 2.3 and later).
The unknown_address_reject_code parameter specifies the response code for rejected requests (default: 450). The response is always 450 in case of a temporary DNS error.

reject_unlisted_sender
Reject the request when the MAIL FROM address is not listed in the list of valid recipients for its domain class. See the smtpd_reject_unlisted_sender parameter description for details. This feature is available in Postfix 2.1 and later.

reject_unverified_sender
Reject the request when mail to the MAIL FROM address is known to bounce, or when the sender address destination is not reachable. Address verification information is managed by the **verify**(8) server; see the ADDRESS_VERIFICATION_README file for details.
The unverified_sender_reject_code parameter specifies the response when an address is known to bounce (default: 450, change into 550 when you are confident that it is safe to do so). Postfix replies with 450 when an address probe failed due to a temporary problem. This feature is

available in Postfix 2.1 and later.

Other restrictions that are valid in this context:

- Generic restrictions that can be used in any SMTP command context, described under smtpd_client_restrictions.
- SMTP command specific restrictions described under smtpd_client_restrictions and smtpd_helo_restrictions.
- SMTP command specific restrictions described under smtpd_recipient_restrictions. When recipient restrictions are listed under smtpd_sender_restrictions, they have effect only with "smtpd_delay_reject = yes", so that $smtpd_sender_restrictions is evaluated at the time of the RCPT TO command.

Examples:

```
smtpd_sender_restrictions = reject_unknown_sender_domain
smtpd_sender_restrictions = reject_unknown_sender_domain,
    check_sender_access hash:/etc/postfix/access
```

smtpd_soft_error_limit (default: 10)

The number of errors a remote SMTP client is allowed to make without delivering mail before the Postfix SMTP server slows down all its responses.

- With Postfix version 2.1 and later, the Postfix SMTP server delays all responses by $smtpd_error_sleep_time seconds.
- With Postfix versions 2.0 and earlier, the Postfix SMTP server delays all responses by (number of errors) seconds.

smtpd_starttls_timeout (default: 300s)

The time limit for Postfix SMTP server write and read operations during TLS startup and shutdown handshake procedures.

This feature is available in Postfix 2.2 and later.

smtpd_timeout (default: 300s)

The time limit for sending a Postfix SMTP server response and for receiving a remote SMTP client request.

Note: if you set SMTP time limits to very large values you may have to update the global ipc_timeout parameter.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

smtpd_tls_CAfile (default: empty)

The file with the certificate of the certification authority (CA) that issued the Postfix SMTP server certificate. This is needed only when the CA certificate is not already present in the server certificate file. This file may also contain the CA certificates of other trusted CAs. You must use this file for the list of trusted CAs if you want to use chroot-mode.

Example:

```
smtpd_tls_CAfile = /etc/postfix/CAcert.pem
```

This feature is available in Postfix 2.2 and later.

smtpd_tls_CApath (default: empty)

Directory with PEM format certificate authority certificates that the Postfix SMTP server offers to remote SMTP clients for the purpose of client certificate verification. Do not forget to create the necessary "hash" links with, for example, "$OPENSSL_HOME/bin/c_rehash /etc/postfix/certs".

To use this option in chroot mode, this directory (or a copy) must be inside the chroot jail. Please note that in this case the CA certificates are not offered to the client, so that e.g. Netscape clients might not offer certificates issued by them. Use of this feature is therefore not recommended.

Example:

```
smtpd_tls_CApath = /etc/postfix/certs
```

This feature is available in Postfix 2.2 and later.

smtpd_tls_always_issue_session_ids (default: yes)
Force the Postfix SMTP server to issue a TLS session id, even when TLS session caching is turned off (smtpd_tls_session_cache_database is empty). This behavior is compatible with Postfix < 2.3.

With Postfix 2.3 and later the Postfix SMTP server can disable session id generation when TLS session caching is turned off. This keeps clients from caching sessions that almost certainly cannot be re-used.

By default, the Postfix SMTP server always generates TLS session ids. This works around a known defect in mail client applications such as MS Outlook, and may also prevent interoperability issues with other MTAs.

Example:

```
smtpd_tls_always_issue_session_ids = no
```

This feature is available in Postfix 2.3 and later.

smtpd_tls_ask_ccert (default: no)
Ask a remote SMTP client for a client certificate. This information is needed for certificate based mail relaying with, for example, the permit_tls_clientcerts feature.

Some clients such as Netscape will either complain if no certificate is available (for the list of CAs in $smtpd_tls_CAfile) or will offer multiple client certificates to choose from. This may be annoying, so this option is "off" by default.

This feature is available in Postfix 2.2 and later.

smtpd_tls_auth_only (default: no)
When TLS encryption is optional in the Postfix SMTP server, do not announce or accept SASL authentication over unencrypted connections.

This feature is available in Postfix 2.2 and later.

smtpd_tls_ccert_verifydepth (default: 5)
The verification depth for remote SMTP client certificates. A depth of 1 is sufficient if the issuing CA is listed in a local CA file. The default value should also suffice for longer chains (the root CA issues special CA which then issues the actual certificate...).

This feature is available in Postfix 2.2 and later.

smtpd_tls_cert_file (default: empty)
File with the Postfix SMTP server RSA certificate in PEM format. This file may also contain the Postfix SMTP server private RSA key.

Public Internet MX hosts without certificates signed by a "reputable" CA must generate, and be prepared to present to most clients, a self-signed or private-CA signed certificate. The client will not be able to authenticate the server, but unless it is running Postfix 2.3 or similar software, it will still insist on a server certificate.

For servers that are **not** public Internet MX hosts, Postfix 2.3 supports configurations with no certificates. This entails the use of just the anonymous TLS ciphers, which are not supported by typical SMTP clients. Since such clients will not, as a rule, fall back to plain text after a TLS handshake failure, the server will be unable to receive email from TLS enabled clients. To avoid accidental configurations with no certificates, Postfix 2.3 enables certificate-less operation only when the administrator explicitly sets "smtpd_tls_cert_file = none". This ensures that new Postfix configurations will not accidentally run with no certificates.

Both RSA and DSA certificates are supported. When both types are present, the cipher used determines which certificate will be presented to the client. For Netscape and OpenSSL clients without special cipher choices the RSA certificate is preferred.

In order to verify a certificate, the CA certificate (in case of a certificate chain, all CA certificates) must be available. You should add these certificates to the server certificate, the server certificate first, then the

issuing CA(s).

Example: the certificate for "server.dom.ain" was issued by "intermediate CA" which itself has a certificate of "root CA". Create the server.pem file with "cat server_cert.pem intermediate_CA.pem root_CA.pem > server.pem".

If you also want to verify client certificates issued by these CAs, you can add the CA certificates to the smtpd_tls_CAfile, in which case it is not necessary to have them in the smtpd_tls_cert_file or smtpd_tls_dcert_file.

A certificate supplied here must be usable as an SSL server certificate and hence pass the "openssl verify -purpose sslserver ..." test.

Example:

```
smtpd_tls_cert_file = /etc/postfix/server.pem
```

This feature is available in Postfix 2.2 and later.

smtpd_tls_cipherlist (default: empty)

Obsolete Postfix < 2.3 control for the Postfix SMTP server TLS cipher list. It is easy to create inter-operability problems by choosing a non-default cipher list. Do not use a non-default TLS cipherlist for MX hosts on the public Internet. Clients that begin the TLS handshake, but are unable to agree on a common cipher, may not be able to send any email to the SMTP server. Using a restricted cipher list may be more appropriate for a dedicated MSA or an internal mailhub, where one can exert some control over the TLS software and settings of the connecting clients.

Note: do not use "" quotes around the parameter value.

This feature is available with Postfix version 2.2. It is not used with Postfix 2.3 and later; use smtpd_tls_mandatory_ciphers instead.

smtpd_tls_dcert_file (default: empty)

File with the Postfix SMTP server DSA certificate in PEM format. This file may also contain the Postfix SMTP server private key.

See the discussion under smtpd_tls_cert_file for more details.

Example:

```
smtpd_tls_dcert_file = /etc/postfix/server-dsa.pem
```

This feature is available in Postfix 2.2 and later.

smtpd_tls_dh1024_param_file (default: empty)

File with DH parameters that the Postfix SMTP server should use with EDH ciphers.

Instead of using the exact same parameter sets as distributed with other TLS packages, it is more secure to generate your own set of parameters with something like the following command:

```
openssl gendh -out /etc/postfix/dh_1024.pem -2 -rand /var/run/egd-pool 1024
```

Your actual source for entropy may differ. Some systems have /dev/random; on other system you may consider using the "Entropy Gathering Daemon EGD", available at http://egd.sourceforge.net/

Example:

```
smtpd_tls_dh1024_param_file = /etc/postfix/dh_1024.pem
```

This feature is available with Postfix version 2.2.

smtpd_tls_dh512_param_file (default: empty)

File with DH parameters that the Postfix SMTP server should use with EDH ciphers.

See also the discussion under the smtpd_tls_dh1024_param_file configuration parameter.

Example:

```
smtpd_tls_dh512_param_file = /etc/postfix/dh_512.pem
```

This feature is available with Postfix version 2.2.

smtpd_tls_dkey_file (default: $smtpd_tls_dcert_file)

File with the Postfix SMTP server DSA private key in PEM format. This file may be combined with the Postfix SMTP server DSA certificate file specified with $smtpd_tls_dcert_file.

The private key must be accessible without a pass-phrase, i.e. it must not be encrypted, but file permissions should grant read/write access only to the system superuser account ("root").

This feature is available in Postfix 2.2 and later.

smtpd_tls_exclude_ciphers (default: empty)

List of ciphers or cipher types to exclude from the SMTP server cipher list at all TLS security levels. Excluding valid ciphers can create interoperability problems. DO NOT exclude ciphers unless it is essential to do so. This is not an OpenSSL cipherlist; it is a simple list separated by whitespace and/or commas. The elements are a single cipher, or one or more "+" separated cipher properties, in which case only ciphers matching **all** the properties are excluded.

Examples (some of these will cause problems):

```
smtpd_tls_exclude_ciphers = aNULL
smtpd_tls_exclude_ciphers = MD5, DES
smtpd_tls_exclude_ciphers = DES+MD5
smtpd_tls_exclude_ciphers = AES256-SHA, DES-CBC3-MD5
smtpd_tls_exclude_ciphers = kEDH+aRSA
```

The first setting disables anonymous ciphers. The next setting disables ciphers that use the MD5 digest algorithm or the (single) DES encryption algorithm. The next setting disables ciphers that use MD5 and DES together. The next setting disables the two ciphers "AES256-SHA" and "DES-CBC3-MD5". The last setting disables ciphers that use "EDH" key exchange with RSA authentication.

This feature is available in Postfix 2.3 and later.

smtpd_tls_key_file (default: $smtpd_tls_cert_file)

File with the Postfix SMTP server RSA private key in PEM format. This file may be combined with the Postfix SMTP server certificate file specified with $smtpd_tls_cert_file.

The private key must be accessible without a pass-phrase, i.e. it must not be encrypted, but file permissions should grant read/write access only to the system superuser account ("root").

smtpd_tls_loglevel (default: 0)

Enable additional Postfix SMTP server logging of TLS activity. Each logging level also includes the information that is logged at a lower logging level.

0 Disable logging of TLS activity.

1 Log TLS handshake and certificate information.

2 Log levels during TLS negotiation.

3 Log hexadecimal and ASCII dump of TLS negotiation process.

4 Also log hexadecimal and ASCII dump of complete transmission after STARTTLS.

Use "smtpd_tls_loglevel = 3" only in case of problems. Use of loglevel 4 is strongly discouraged.

This feature is available in Postfix 2.2 and later.

smtpd_tls_mandatory_ciphers (default: medium)

The minimum TLS cipher grade that the Postfix SMTP server will use with mandatory TLS encryption. Cipher types listed in smtpd_tls_mandatory_exclude_ciphers or smtpd_tls_exclude_ciphers are excluded from the base definition of the selected cipher grade. With opportunistic TLS encryption, the "export" grade is used unconditionally with exclusions specified only via smtpd_tls_exclude_ciphers.

The following cipher grades are supported:

export Enable the mainstream "EXPORT" grade or better OpenSSL ciphers. This is the most appropriate setting for public MX hosts, and is always used with opportunistic TLS encryption. The underlying cipherlist is specified via the tls_export_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_export_cipherlist includes anonymous ciphers, but these are automatically filtered out if the server is configured to ask for client certificates. If you must always exclude anonymous ciphers, set "smtpd_tls_exclude_ciphers = aNULL". To exclude anonymous ciphers only when TLS is enforced, set "smtpd_tls_mandatory_exclude_ciphers = aNULL".

low Enable the mainstream "LOW" grade or better OpenSSL ciphers. The underlying cipherlist is specified via the tls_low_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_low_cipherlist includes anonymous ciphers, but these are automatically filtered out if the server is configured to ask for client certificates. If you must always exclude anonymous ciphers, set "smtpd_tls_exclude_ciphers = aNULL". To exclude anonymous ciphers only when TLS is enforced, set "smtpd_tls_mandatory_exclude_ciphers = aNULL".

medium
Enable the mainstream "MEDIUM" grade or better OpenSSL ciphers. These are essentially the 128-bit or stronger ciphers. This is the default minimum strength for mandatory TLS encryption. MSAs that enforce TLS and have clients that do not support any "MEDIUM" or "HIGH" grade ciphers, may need to configure a weaker ("low" or "export") minimum cipher grade. The underlying cipherlist is specified via the tls_medium_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_medium_cipherlist includes anonymous ciphers, but these are automatically filtered out if the server is configured to ask for client certificates. If you must always exclude anonymous ciphers, set "smtpd_tls_exclude_ciphers = aNULL". To exclude anonymous ciphers only when TLS is enforced, set "smtpd_tls_mandatory_exclude_ciphers = aNULL".

high Enable only the mainstream "HIGH" grade OpenSSL ciphers. The underlying cipherlist is specified via the tls_high_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_high_cipherlist includes anonymous ciphers, but these are automatically filtered out if the server is configured to ask for client certificates. If you must always exclude anonymous ciphers, set "smtpd_tls_exclude_ciphers = aNULL". To exclude anonymous ciphers only when TLS is enforced, set "smtpd_tls_mandatory_exclude_ciphers = aNULL".

null Enable only the "NULL" OpenSSL ciphers, these provide authentication without encryption. This setting is only appropriate in the rare case that all clients are prepared to use NULL ciphers (not normally enabled in TLS clients). The underlying cipherlist is specified via the tls_null_cipherlist configuration parameter, which you are strongly encouraged to not change. The default value of tls_null_cipherlist excludes anonymous ciphers (OpenSSL 0.9.8 has NULL ciphers that offer data integrity without encryption or authentication).

This feature is available in Postfix 2.3 and later.

smtpd_tls_mandatory_exclude_ciphers (default: empty)
Additional list of ciphers or cipher types to exclude from the SMTP server cipher list at mandatory TLS security levels. This list works in addition to the exclusions listed with smtpd_tls_exclude_ciphers (see there for syntax details).

This feature is available in Postfix 2.3 and later.

smtpd_tls_mandatory_protocols (default: SSLv3, TLSv1)
The TLS protocols accepted by the Postfix SMTP server with mandatory TLS encryption. With opportunistic TLS encryption, all protocols are always accepted. If the list is empty, the server supports all available TLS protocol versions. A non-empty value is a list of protocol names separated by whitespace, commas or colons. The supported protocol names are "SSLv2", "SSLv3" and "TLSv1", and are not case sensitive.

Example:

```
smtpd_tls_mandatory_protocols = SSLv3, TLSv1
```

This feature is available in Postfix 2.3 and later.

smtpd_tls_received_header (default: no)

Request that the Postfix SMTP server produces Received: message headers that include information about the protocol and cipher used, as well as the client CommonName and client certificate issuer CommonName. This is disabled by default, as the information may be modified in transit through other mail servers. Only information that was recorded by the final destination can be trusted.

This feature is available in Postfix 2.2 and later.

smtpd_tls_req_ccert (default: no)

With mandatory TLS encryption, require a remote SMTP client certificate in order to allow TLS connections to proceed. This option implies "smtpd_tls_ask_ccert = yes".

When TLS encryption is optional, this setting is ignored with a warning written to the mail log.

This feature is available in Postfix 2.2 and later.

smtpd_tls_security_level (default: empty)

The SMTP TLS security level for the Postfix SMTP server; when a non-empty value is specified, this overrides the obsolete parameters smtpd_use_tls and smtpd_enforce_tls. This parameter is ignored with "smtpd_tls_wrappermode = yes".

Specify one of the following security levels:

none TLS will not be used.

may Opportunistic TLS: announce STARTTLS support to SMTP clients, but do not require that clients use TLS encryption.

encrypt
Mandatory TLS encryption: announce STARTTLS support to SMTP clients, and require that clients use TLS encryption. According to RFC 2487 this MUST NOT be applied in case of a publicly-referenced SMTP server. Instead, this option should be used only on dedicated servers.

Note 1: the "verify" and "secure" levels are not supported. The Postfix SMTP server logs a warning and uses "encrypt" instead. To verify SMTP client certificates, see TLS_README for a discussion of the smtpd_tls_ask_ccert, smtpd_tls_req_ccert, and permit_tls_clientcerts features.

Note 2: The parameter setting "smtpd_tls_security_level = encrypt" implies "smtpd_tls_auth_only = yes".

Note 3: when invoked via "sendmail -bs", Postfix will never offer STARTTLS due to insufficient privileges to access the server private key. This is intended behavior.

This feature is available in Postfix 2.3 and later.

smtpd_tls_session_cache_database (default: empty)

Name of the file containing the optional Postfix SMTP server TLS session cache. Specify a database type that supports enumeration, such as **btree** or **sdbm**; there is no need to support concurrent access. The file is created if it does not exist. The **smtpd**(8) daemon does not use this parameter directly, rather the cache is implemented indirectly in the **tlsmgr**(8) daemon. This means that per-smtpd-instance master.cf overrides of this parameter are not effective. Note, that each of the cache databases supported by **tlsmgr**(8) daemon: $smtpd_tls_session_cache_database, $smtp_tls_session_cache_database (and with Postfix 2.3 and later $lmtp_session_cache_database), needs to be stored separately, it is not at this time possible to store multiple caches in a single database.

Note: **dbm** databases are not suitable. TLS session objects are too large.

Example:

```
smtpd_tls_session_cache_database = btree:/var/spool/postfix/smtpd_scache
```

This feature is available in Postfix 2.2 and later.

smtpd_tls_session_cache_timeout (default: 3600s)
The expiration time of Postfix SMTP server TLS session cache information. A cache cleanup is performed periodically every $smtpd_tls_session_cache_timeout seconds. As with $smtpd_tls_session_cache_database, this parameter is implemented in the **tlsmgr**(8) daemon and therefore per-smtpd-instance master.cf overrides are not possible.

This feature is available in Postfix 2.2 and later.

smtpd_tls_wrappermode (default: no)
Run the Postfix SMTP server in the non-standard "wrapper" mode, instead of using the STARTTLS command.

If you want to support this service, enable a special port in master.cf, and specify "-o smtpd_tls_wrappermode=yes" on the SMTP server's command line. Port 465 (smtps) was once chosen for this purpose.

This feature is available in Postfix 2.2 and later.

smtpd_use_tls (default: no)
Opportunistic TLS: announce STARTTLS support to SMTP clients, but do not require that clients use TLS encryption.

Note: when invoked via "**sendmail -bs**", Postfix will never offer STARTTLS due to insufficient privileges to access the server private key. This is intended behavior.

This feature is available in Postfix 2.2 and later. With Postfix 2.3 and later use smtpd_tls_security_level instead.

soft_bounce (default: no)
Safety net to keep mail queued that would otherwise be returned to the sender. This parameter disables locally-generated bounces, and prevents the Postfix SMTP server from rejecting mail permanently, by changing 5xx reply codes into 4xx. However, soft_bounce is no cure for address rewriting mistakes or mail routing mistakes.

Example:

```
soft_bounce = yes
```

stale_lock_time (default: 500s)
The time after which a stale exclusive mailbox lockfile is removed. This is used for delivery to file or mailbox.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

strict_7bit_headers (default: no)
Reject mail with 8-bit text in message headers. This blocks mail from poorly written applications.

This feature should not be enabled on a general purpose mail server, because it is likely to reject legitimate email.

This feature is available in Postfix 2.0 and later.

strict_8bitmime (default: no)
Enable both strict_7bit_headers and strict_8bitmime_body.

This feature should not be enabled on a general purpose mail server, because it is likely to reject legitimate email.

This feature is available in Postfix 2.0 and later.

strict_8bitmime_body (default: no)
Reject 8-bit message body text without 8-bit MIME content encoding information. This blocks mail from poorly written applications.

Unfortunately, this also rejects majordomo approval requests when the included request contains valid 8-bit MIME mail, and it rejects bounces from mailers that do not MIME encapsulate 8-bit content (for example, bounces from qmail or from old versions of Postfix).

This feature should not be enabled on a general purpose mail server, because it is likely to reject legitimate email.

This feature is available in Postfix 2.0 and later.

strict_mime_encoding_domain (default: no)
Reject mail with invalid Content-Transfer-Encoding: information for the message/* or multipart/* MIME content types. This blocks mail from poorly written software.

This feature should not be enabled on a general purpose mail server, because it will reject mail after a single violation.

This feature is available in Postfix 2.0 and later.

strict_rfc821_envelopes (default: no)
Require that addresses received in SMTP MAIL FROM and RCPT TO commands are enclosed with <>, and that those addresses do not contain RFC 822 style comments or phrases. This stops mail from poorly written software.

By default, the Postfix SMTP server accepts RFC 822 syntax in MAIL FROM and RCPT TO addresses.

sun_mailtool_compatibility (default: no)
Obsolete SUN mailtool compatibility feature. Instead, use "mailbox_delivery_lock = dotlock".

swap_bangpath (default: yes)
Enable the rewriting of "site!user" into "user@site". This is necessary if your machine is connected to UUCP networks. It is enabled by default.

Note: with Postfix version 2.2, message header address rewriting happens only when one of the following conditions is true:

- The message is received with the Postfix **sendmail**(1) command,
- The message is received from a network client that matches $local_header_rewrite_clients,
- The message is received from the network, and the remote_header_rewrite_domain parameter specifies a non-empty value.

To get the behavior before Postfix version 2.2, specify "local_header_rewrite_clients = static:all".

Example:

```
swap_bangpath = no
```

syslog_facility (default: mail)
The syslog facility of Postfix logging. Specify a facility as defined in syslog.**conf**(5). The default facility is "mail".

Warning: a non-default syslog_facility setting takes effect only after a Postfix process has completed initialization. Errors during process initialization will be logged with the default facility. Examples are errors while parsing the command line arguments, and errors while accessing the Postfix main.cf configuration file.

syslog_name (default: postfix)
The mail system name that is prepended to the process name in syslog records, so that "smtpd" becomes, for example, "postfix/smtpd".

Warning: a non-default syslog_name setting takes effect only after a Postfix process has completed initialization. Errors during process initialization will be logged with the default name. Examples are errors while parsing the command line arguments, and errors while accessing the Postfix main.cf configuration file.

tls_daemon_random_bytes (default: 32)
The number of pseudo-random bytes that an **smtp**(8) or **smtpd**(8) process requests from the **tlsmgr**(8) server in order to seed its internal pseudo random number generator (PRNG). The default of 32 bytes (equivalent to 256 bits) is sufficient to generate a 128bit (or 168bit) session key.

This feature is available in Postfix 2.2 and later.

tls_export_cipherlist (default: ALL:+RC4:@STRENGTH)
The OpenSSL cipherlist for "EXPORT" or higher grade ciphers. This defines the meaning of the "export" setting in smtpd_tls_mandatory_ciphers, smtp_tls_mandatory_ciphers and lmtp_tls_mandatory_ciphers. This is the cipherlist for the opportunistic ("may") TLS client security level and is the default cipherlist for the SMTP server. You are strongly encouraged to not change this setting.

This feature is available in Postfix 2.3 and later.

tls_high_cipherlist (default: ALL:!EXPORT:!LOW:!MEDIUM:+RC4:@STRENGTH)
The OpenSSL cipherlist for "HIGH" grade ciphers. This defines the meaning of the "high" setting in smtpd_tls_mandatory_ciphers, smtp_tls_mandatory_ciphers and lmtp_tls_mandatory_ciphers. You are strongly encouraged to not change this setting.

This feature is available in Postfix 2.3 and later.

tls_low_cipherlist (default: ALL:!EXPORT:+RC4:@STRENGTH)
The OpenSSL cipherlist for "LOW" or higher grade ciphers. This defines the meaning of the "low" setting in smtpd_tls_mandatory_ciphers, smtp_tls_mandatory_ciphers and lmtp_tls_mandatory_ciphers. You are strongly encouraged to not change this setting.

This feature is available in Postfix 2.3 and later.

tls_medium_cipherlist (default: ALL:!EXPORT:!LOW:+RC4:@STRENGTH)
The OpenSSL cipherlist for "MEDIUM" or higher grade ciphers. This defines the meaning of the "medium" setting in smtpd_tls_mandatory_ciphers, smtp_tls_mandatory_ciphers and lmtp_tls_mandatory_ciphers. This is the default cipherlist for mandatory TLS encryption in the TLS client (with anonymous ciphers disabled when verifying server certificates). You are strongly encouraged to not change this setting.

This feature is available in Postfix 2.3 and later.

tls_null_cipherlist (default: eNULL:!aNULL)
The OpenSSL cipherlist for "NULL" grade ciphers that provide authentication without encryption. This defines the meaning of the "null" setting in smtpd_mandatory_tls_ciphers, smtp_tls_mandatory_ciphers and lmtp_tls_mandatory_ciphers. You are strongly encouraged to not change this setting.

This feature is available in Postfix 2.3 and later.

tls_random_bytes (default: 32)
The number of bytes that **tlsmgr**(8) reads from $tls_random_source when (re)seeding the in-memory pseudo random number generator (PRNG) pool. The default of 32 bytes (256 bits) is good enough for 128bit symmetric keys. If using EGD or a device file, a maximum of 255 bytes is read.

This feature is available in Postfix 2.2 and later.

tls_random_exchange_name (default: ${config_directory}/prng_exch)
Name of the pseudo random number generator (PRNG) state file that is maintained by **tlsmgr**(8). The file is created when it does not exist, and its length is fixed at 1024 bytes.

Since this file is modified by Postfix, it should probably be kept in the /var file system, instead of under $config_directory. The location should not be inside the chroot jail.

This feature is available in Postfix 2.2 and later.

tls_random_prng_update_period (default: 3600s)
The time between attempts by **tlsmgr**(8) to save the state of the pseudo random number generator (PRNG) to the file specified with $tls_random_exchange_name.

This feature is available in Postfix 2.2 and later.

tls_random_reseed_period (default: 3600s)
The maximal time between attempts by **tlsmgr**(8) to re-seed the in-memory pseudo random number generator (PRNG) pool from external sources. The actual time between re-seeding attempts is calculated using the PRNG, and is between 0 and the time specified.

This feature is available in Postfix 2.2 and later.

tls_random_source (default: see postconf -d output)

The external entropy source for the in-memory **tlsmgr**(8) pseudo random number generator (PRNG) pool. Be sure to specify a non-blocking source. If this source is not a regular file, the entropy source type must be prepended: egd:/path/to/egd_socket for a source with EGD compatible socket interface, or dev:/path/to/device for a device file.

Note: on OpenBSD systems specify /dev/arandom when /dev/urandom gives timeout errors.

This feature is available in Postfix 2.2 and later.

trace_service_name (default: trace)

The name of the trace service. This service is implemented by the **bounce**(8) daemon and maintains a record of mail deliveries and produces a mail delivery report when verbose delivery is requested with "**sendmail -v**".

This feature is available in Postfix 2.1 and later.

transport_maps (default: empty)

Optional lookup tables with mappings from recipient address to (message delivery transport, next-hop destination). See **transport**(5) for details.

Specify zero or more "type:table" lookup tables. If you use this feature with local files, run "**postmap /etc/postfix/transport**" after making a change.

For safety reasons, as of Postfix 2.3 this feature does not allow $number substitutions in regular expression maps.

Examples:

```
transport_maps = dbm:/etc/postfix/transport
transport_maps = hash:/etc/postfix/transport
```

transport_retry_time (default: 60s)

The time between attempts by the Postfix queue manager to contact a malfunctioning message delivery transport.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

trigger_timeout (default: 10s)

The time limit for sending a trigger to a Postfix daemon (for example, the **pickup**(8) or **qmgr**(8) daemon). This time limit prevents programs from getting stuck when the mail system is under heavy load.

Time units: s (seconds), m (minutes), h (hours), d (days), w (weeks). The default time unit is s (seconds).

undisclosed_recipients_header (default: To: undisclosed-recipients:;)

Message header that the Postfix **cleanup**(8) server inserts when a message contains no To: or Cc: message header.

unknown_address_reject_code (default: 450)

The numerical Postfix SMTP server response code when a sender or recipient address is rejected by the reject_unknown_sender_domain or reject_unknown_recipient_domain restriction. The response is always 450 in case of a temporary DNS error.

Do not change this unless you have a complete understanding of RFC 821.

unknown_client_reject_code (default: 450)

The numerical Postfix SMTP server response code when a client without valid address <=> name mapping is rejected by the reject_unknown_client_hostname restriction. The SMTP server always replies with 450 when the mapping failed due to a temporary error condition.

Do not change this unless you have a complete understanding of RFC 821.

unknown_hostname_reject_code (default: 450)
The numerical Postfix SMTP server response code when the hostname specified with the HELO or EHLO command is rejected by the reject_unknown_helo_hostname restriction.

Do not change this unless you have a complete understanding of RFC 821.

unknown_local_recipient_reject_code (default: 550)
The numerical Postfix SMTP server response code when a recipient address is local, and $local_recipient_maps specifies a list of lookup tables that does not match the recipient. A recipient address is local when its domain matches $mydestination, $proxy_interfaces or $inet_interfaces.

The default setting is 550 (reject mail) but it is safer to initially use 450 (try again later) so you have time to find out if your local_recipient_maps settings are OK.

Example:

```
unknown_local_recipient_reject_code = 450
```

This feature is available in Postfix 2.0 and later.

unknown_relay_recipient_reject_code (default: 550)
The numerical Postfix SMTP server reply code when a recipient address matches $relay_domains, and relay_recipient_maps specifies a list of lookup tables that does not match the recipient address.

This feature is available in Postfix 2.0 and later.

unknown_virtual_alias_reject_code (default: 550)
The SMTP server reply code when a recipient address matches $virtual_alias_domains, and $virtual_alias_maps specifies a list of lookup tables that does not match the recipient address.

This feature is available in Postfix 2.0 and later.

unknown_virtual_mailbox_reject_code (default: 550)
The SMTP server reply code when a recipient address matches $virtual_mailbox_domains, and $virtual_mailbox_maps specifies a list of lookup tables that does not match the recipient address.

This feature is available in Postfix 2.0 and later.

unverified_recipient_reject_code (default: 450)
The numerical Postfix SMTP server response when a recipient address is rejected by the reject_unverified_recipient restriction.

Unlike elsewhere in Postfix, you can specify 250 in order to accept the address anyway.

Do not change this unless you have a complete understanding of RFC 821.

This feature is available in Postfix 2.1 and later.

unverified_sender_reject_code (default: 450)
The numerical Postfix SMTP server response code when a recipient address is rejected by the reject_unverified_sender restriction.

Unlike elsewhere in Postfix, you can specify 250 in order to accept the address anyway.

Do not change this unless you have a complete understanding of RFC 821.

This feature is available in Postfix 2.1 and later.

verp_delimiter_filter (default: -=+)
The characters Postfix accepts as VERP delimiter characters on the Postfix **sendmail**(1) command line and in SMTP commands.

This feature is available in Postfix 1.1 and later.

virtual_alias_domains (default: $virtual_alias_maps)
Postfix is final destination for the specified list of virtual alias domains, that is, domains for which all addresses are aliased to addresses in other local or remote domains. The SMTP server validates recipient addresses with $virtual_alias_maps and rejects non-existent recipients. See also the virtual alias domain

class in the ADDRESS_CLASS_README file

This feature is available in Postfix 2.0 and later. The default value is backwards compatible with Postfix version 1.1.

The default value is $virtual_alias_maps so that you can keep all information about virtual alias domains in one place. If you have many users, it is better to separate information that changes more frequently (virtual address -> local or remote address mapping) from information that changes less frequently (the list of virtual domain names).

Specify a list of host or domain names, "/file/name" or "type:table" patterns, separated by commas and/or whitespace. A "/file/name" pattern is replaced by its contents; a "type:table" lookup table is matched when a table entry matches a lookup string (the lookup result is ignored). Continue long lines by starting the next line with whitespace. Specify "!pattern" to exclude a host or domain name from the list. The form "!/file/name" is supported only in Postfix version 2.4 and later.

See also the VIRTUAL_README and ADDRESS_CLASS_README documents for further information.

Example:

```
virtual_alias_domains = virtual1.tld virtual2.tld
```

virtual_alias_expansion_limit (default: 1000)

The maximal number of addresses that virtual alias expansion produces from each original recipient.

This feature is available in Postfix 2.1 and later.

virtual_alias_maps (default: $virtual_maps)

Optional lookup tables that alias specific mail addresses or domains to other local or remote address. The table format and lookups are documented in **virtual**(5). For an overview of Postfix address manipulations see the ADDRESS_REWRITING_README document.

This feature is available in Postfix 2.0 and later. The default value is backwards compatible with Postfix version 1.1.

If you use this feature with indexed files, run "**postmap /etc/postfix/virtual**" after changing the file.

Examples:

```
virtual_alias_maps = dbm:/etc/postfix/virtual
virtual_alias_maps = hash:/etc/postfix/virtual
```

virtual_alias_recursion_limit (default: 1000)

The maximal nesting depth of virtual alias expansion. Currently the recursion limit is applied only to the left branch of the expansion graph, so the depth of the tree can in the worst case reach the sum of the expansion and recursion limits. This may change in the future.

This feature is available in Postfix 2.1 and later.

virtual_destination_concurrency_limit (default: $default_destination_concurrency_limit)

The maximal number of parallel deliveries to the same destination via the virtual message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

virtual_destination_recipient_limit (default: $default_destination_recipient_limit)

The maximal number of recipients per delivery via the virtual message delivery transport. This limit is enforced by the queue manager. The message delivery transport name is the first field in the entry in the master.cf file.

Setting this parameter to a value of 1 changes the meaning of virtual_destination_concurrency_limit from concurrency per domain into concurrency per recipient.

virtual_gid_maps (default: empty)

Lookup tables with the per-recipient group ID for **virtual**(8) mailbox delivery.

In a lookup table, specify a left-hand side of "@domain.tld" to match any user in the specified domain that

does not have a specific "user@domain.tld" entry.

When a recipient address has an optional address extension (user+foo@domain.tld), the **virtual**(8) delivery agent looks up the full address first, and when the lookup fails, it looks up the unextended address (user@domain.tld).

Note 1: for security reasons, the **virtual**(8) delivery agent disallows regular expression substitution of $1 etc. in regular expression lookup tables, because that would open a security hole.

Note 2: for security reasons, the **virtual**(8) delivery agent will silently ignore requests to use the **proxymap**(8) server. Instead it will open the table directly. Before Postfix version 2.2, the **virtual**(8) delivery agent will terminate with a fatal error.

virtual_mailbox_base (default: empty)

A prefix that the **virtual**(8) delivery agent prepends to all pathname results from $virtual_mailbox_maps table lookups. This is a safety measure to ensure that an out of control map doesn't litter the file system with mailboxes. While virtual_mailbox_base could be set to "/", this setting isn't recommended.

Example:

```
virtual_mailbox_base = /var/mail
```

virtual_mailbox_domains (default: $virtual_mailbox_maps)

Postfix is final destination for the specified list of domains; mail is delivered via the $virtual_transport mail delivery transport. By default this is the Postfix **virtual**(8) delivery agent. The SMTP server validates recipient addresses with $virtual_mailbox_maps and rejects mail for non-existent recipients. See also the virtual mailbox domain class in the ADDRESS_CLASS_README file.

This parameter expects the same syntax as the mydestination configuration parameter.

This feature is available in Postfix 2.0 and later. The default value is backwards compatible with Postfix version 1.1.

virtual_mailbox_limit (default: 51200000)

The maximal size in bytes of an individual mailbox or maildir file, or zero (no limit).

virtual_mailbox_lock (default: see postconf -d output)

How to lock a UNIX-style **virtual**(8) mailbox before attempting delivery. For a list of available file locking methods, use the "**postconf -l**" command.

This setting is ignored with **maildir** style delivery, because such deliveries are safe without application-level locks.

Note 1: the **dotlock** method requires that the recipient UID or GID has write access to the parent directory of the recipient's mailbox file.

Note 2: the default setting of this parameter is system dependent.

virtual_mailbox_maps (default: empty)

Optional lookup tables with all valid addresses in the domains that match $virtual_mailbox_domains.

In a lookup table, specify a left-hand side of "@domain.tld" to match any user in the specified domain that does not have a specific "user@domain.tld" entry.

The **virtual**(8) delivery agent uses this table to look up the per-recipient mailbox or maildir pathname. If the lookup result ends in a slash ("/"), maildir-style delivery is carried out, otherwise the path is assumed to specify a UNIX-style mailbox file. Note that $virtual_mailbox_base is unconditionally prepended to this path.

When a recipient address has an optional address extension (user+foo@domain.tld), the **virtual**(8) delivery agent looks up the full address first, and when the lookup fails, it looks up the unextended address (user@domain.tld).

Note 1: for security reasons, the **virtual**(8) delivery agent disallows regular expression substitution of $1 etc. in regular expression lookup tables, because that would open a security hole.

Note 2: for security reasons, the **virtual**(8) delivery agent will silently ignore requests to use the **proxymap**(8) server. Instead it will open the table directly. Before Postfix version 2.2, the **virtual**(8) delivery agent will terminate with a fatal error.

virtual_maps (default: empty)

Optional lookup tables with a) names of domains for which all addresses are aliased to addresses in other local or remote domains, and b) addresses that are aliased to addresses in other local or remote domains. Available before Postfix version 2.0. With Postfix version 2.0 and later, this is replaced by separate controls: virtual_alias_domains and virtual_alias_maps.

virtual_minimum_uid (default: 100)

The minimum user ID value that the **virtual**(8) delivery agent accepts as a result from $virtual_uid_maps table lookup. Returned values less than this will be rejected, and the message will be deferred.

virtual_transport (default: virtual)

The default mail delivery transport and next-hop destination for final delivery to domains listed with $virtual_mailbox_domains. This information can be overruled with the **transport**(5) table.

Specify a string of the form *transport:nexthop*, where *transport* is the name of a mail delivery transport defined in master.cf. The *:nexthop* part is optional. For more details see the **transport**(5) manual page.

This feature is available in Postfix 2.0 and later.

virtual_uid_maps (default: empty)

Lookup tables with the per-recipient user ID that the **virtual**(8) delivery agent uses while writing to the recipient's mailbox.

In a lookup table, specify a left-hand side of "@domain.tld" to match any user in the specified domain that does not have a specific "user@domain.tld" entry.

When a recipient address has an optional address extension (user+foo@domain.tld), the **virtual**(8) delivery agent looks up the full address first, and when the lookup fails, it looks up the unextended address (user@domain.tld).

Note 1: for security reasons, the **virtual**(8) delivery agent disallows regular expression substitution of $1 etc. in regular expression lookup tables, because that would open a security hole.

Note 2: for security reasons, the **virtual**(8) delivery agent will silently ignore requests to use the **proxymap**(8) server. Instead it will open the table directly. Before Postfix version 2.2, the **virtual**(8) delivery agent will terminate with a fatal error.

SEE ALSO

postconf(1), Postfix configuration parameter maintenance
master(5), Postfix daemon configuration maintenance

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

printcap — printer capability data base

SYNOPSIS

printcap

DESCRIPTION

The **printcap** data base is a simplified version of the termcap(5) data base used to describe line printers. The spooling system accesses the **printcap** file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base is used to describe one printer. This data base may not be substituted for, as is possible for termcap(5), because it may allow accounting to be bypassed.

The default printer is normally *lp*, though the environment variable PRINTER may be used to override this. Each spooling utility supports an option, **-P** *printer*, to allow explicit naming of a destination printer.

Refer to the *4.3 BSD Line Printer Spooler Manual* for a complete discussion on how to set up the database for a given printer.

CAPABILITIES

Refer to termcap(5) for a description of the file layout.

Name	Type	Default	Description
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl(2) call)
cf	str	NULL	cifplot data filter
df	str	NULL	tex data filter (DVI format)
fc	num	0	if lp is a tty, clear flag bits (sgtty.h)
ff	str	'\f'	string to send for a form feed
fo	bool	false	print a form feed when device is opened
fs	num	0	like 'fc' but set bits
gf	str	NULL	graph data filter (plot(3) format
hl	bool	false	print the burst header page last
ic	bool	false	driver supports (non standard) ioctl to indent printout
if	str	NULL	name of text filter which does accounting
lf	str	/dev/console	error logging file name
lo	str	lock	name of lock file
lp	str	/dev/lp	device name to open for output to local printer, or port@host for remote printer/printer on print server
ms	str	NULL	list of terminal modes to set or clear
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device independent troff)
of	str	NULL	name of output filtering program
pc	num	200	price per foot or page in hundredths of cents
pf	str	NULL	filter for printing PostScript files
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN style text files
rg	str	NULL	restricted group. Only members of group allowed access

rm	str	NULL	machine name for remote printer or port@host for a remote printer on a port other than the standard port. (also suppress the burst page, see **NOTES**)
rp	str	"lp"	remote printer name argument
rs	bool	false	restrict remote users to those with local accounts
rw	bool	false	open the printer device for reading and writing
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies
sd	str	/var/spool/output/lpd	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header (local only, see **NOTES**)
st	str	status	status file name
tf	str	NULL	troff data filter (cat phototypesetter)
tr	str	NULL	trailer string to print when queue empties
vf	str	NULL	raster image filter
xc	num	0	if lp is a tty, clear local mode bits (tty(4))
xs	num	0	like 'xc' but set bits

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

FILTERS

If a printer is specified via **lp** (either local or remote), the lpd(8) daemon creates a pipeline of *filters* to process files for various printer types. The pipeline is not set up for remote printers specified via **rm** unless the local host is the same as the remote printer host given. The filters selected depend on the flags passed to lpr(1). The pipeline set up is:

```
p       pr | if regular text + pr(1)
none    if      regular text
c       cf      cifplot
d       df      DVI (tex)
g       gf      plot(3)
n       nf      ditroff
o       pf      PostScript
f       rf      Fortran
t       tf      troff
v       vf      raster image
```

The **if** filter is invoked with arguments:

if [**-c**] **-w***width* **-l***length* **-i***indent* **-n** *login* **-h** *host acct-file*

The **-c** flag is passed only if the **-l** flag (pass control characters literally) is specified to lpr(1). The *width* and *length* specify the page width and length (from **pw** and **pl** respectively) in characters. The **-n** and **-h** parameters specify the login name and host name of the owner of the job respectively. The *acct-file* option is passed from the **af** **printcap** entry.

If no **if** is specified, **of** is used instead, with the distinction that **of** is opened only once, while **if** is opened for every individual job. Thus, **if** is better suited to performing accounting. The **of** is only given the *width* and *length* flags.

All other filters are called as:

filter -x*width* **-y***length* **-n** *login* **-h** *host acct-file*

where *width* and *length* are represented in pixels, specified by the **px** and **py** entries respectively.

All filters take *stdin* as the file, *stdout* as the printer, may log either to *stderr* or using `syslog`(3), and must not ignore `SIGINT`.

Filters can communicate errors to lpd by their exit code and by modifying the mode of the spool lock file as follows:

Exit code	**Description**
0	Success.
1	An attempt is made to reprint the job and mail is sent if it fails.
2	`lpd`(8) silently discards the job.
n	`lpd`(8) discards the job and mail is sent.
lock code	**Description**
u+x	Stop printing and leave queue disabled (S_IXUSR).
o+x	Rebuild the queue (S_IXOTH).

LOGGING

Error messages generated by the line printer programs themselves (that is, the lp∗ programs) are logged by `syslog`(3) using the `LPR` facility. Messages printed on *stderr* of one of the filters are sent to the corresponding **`lf`** file. The filters may, of course, use `syslog`(3) themselves.

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

SEE ALSO

`lpq`(1), `lpr`(1), `lprm`(1), `termcap`(5), `lpc`(8), `lpd`(8), `pac`(8)

4.3 BSD Line Printer Spooler Manual.

NOTES

The **sh** flag is a function of the spooler with the locally attached printer, and so has no effect when used with **rm**. NetBSD never adds a burst page when used as a remote spooler. To suppress the burst page for other systems or dedicated devices, refer to the documentation for those systems or devices.

HISTORY

The **`printcap`** file format appeared in 4.2BSD.

NAME

protocols — protocol name data base

DESCRIPTION

The **protocols** file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official protocol name
protocol number
aliases

Items are separated by any number of blanks and/or tab characters. A hash ("#") indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

`/etc/protocols` The **protocols** file resides in `/etc`.

SEE ALSO

`getprotoent`(3)

HISTORY

The **protocols** file format appeared in 4.2BSD.

BUGS

A name server should be used instead of a static file.

NAME

racoon.conf — configuration file for racoon

DESCRIPTION

racoon.conf is the configuration file for the racoon(8) ISAKMP daemon. racoon(8) negotiates security associations for itself (ISAKMP SA, or phase 1 SA) and for kernel IPsec (IPsec SA, or phase 2 SA). The file consists of a sequence of directives and statements. Each directive is composed by a tag and statements, enclosed by '{' and '}'. Lines beginning with '#' are comments.

Meta Syntax

Keywords and special characters that the parser expects exactly are displayed using **this** font. Parameters are specified with *this* font. Square brackets ('[' and ']') are used to show optional keywords and parameters. Note that you have to pay attention when this manual is describing *port* numbers. The *port* number is always enclosed by '[' and ']'. In this case, the port number is not an optional keyword. If it is possible to omit the *port* number, the expression becomes [[*port*]]. The vertical bar ('|') is used to indicate a choice between optional parameters. Parentheses ('(' and ')') are used to group keywords and parameters when necessary. Major parameters are listed below.

number means a hexadecimal or a decimal number. The former must be prefixed with 0x.
string
path
file means any string enclosed in '"' (double quotes).
address means IPv6 and/or IPv4 address.
port means a TCP/UDP port number. The port number is always enclosed by '[' and ']'.
timeunit is one of following: **sec**, **secs**, **second**, **seconds**, **min**, **mins**, **minute**, **minutes**, **hour**, **hours**.

Privilege separation

privsep { *statements* }

Specifies privilege separation parameters. When enabled, these enable racoon(8) to operate with an unprivileged instance doing most of the work, while a privileged instance takes care of performing the following operations as root: reading PSK and private keys, launching hook scripts, and validating passwords against system databases or against PAM. Please note that using privilege separation makes changes to the *listen* and *paths* sections ignored upon configuration reloads. A racoon(8) restart is required if you want such changes to be taken into account.

user *user*;
The user to which the unprivileged instance of racoon(8), should switch. This can be a quoted user name or a numeric UID.

group *group*;
The group the unprivileged instance of racoon(8), should switch. This can be a quoted group name or a numeric GID.

chroot *path*;
A directory to which the unprivileged instance of racoon(8) should chroot(2). This directory should hold a tree where the following files must be reachable:
/dev/random
/dev/urandom
The certificates
The file containing the Xauth banner

The PSK file, the private keys, and the hook scripts are accessed through the privileged instance of racoon(8) and do not need to be reachable in the chroot(2)'ed tree.

Path Specification

This section specifies various paths used by racoon. When running in privilege separation mode, **certificate** and **script** paths are mandatory. A racoon(8) restart is required if you want path changes to be taken into account.

path include *path*;
Specifies a path to include a file. See **File Inclusion**.

path pre_shared_key *file*;
Specifies a file containing pre-shared key(s) for various ID(s). See **Pre-shared key File**.

path certificate *path*;
racoon(8) will search this directory if a certificate or certificate request is received. If you run with privilege separation, racoon(8) will refuse to use a certificate stored outside of this directory.

path backupsa *file*;
Specifies a file to which SA information negotiated by racoon should be stored. racoon(8) will install SA(s) from the file when started with the **-B** flag. The file is growing because racoon(8) simply adds SAs to it. You should maintain the file manually.

path script *path*;
racoon(8) will search this directory for scripts hooks. If you run with privilege separation, racoon(8) will refuse to execute a script stored outside of this directory.

path pidfile *file*;
Specifies file where to store PID of process. If path starts with / it is treated as an absolute path. Otherwise, it is treated as a relative path to the VARRUN directory specified at compilation time. Default is racoon.pid.

File Inclusion

include *file*
Specifies other configuration files to be included.

Identifier Specification

is obsolete. It must be defined at each **remote** directive.

Timer Specification

timer { *statements* **}**
This section specifies various timer values used by racoon.

counter *number*;
The maximum number of retries to send. The default is 5.

interval *number timeunit*;
The interval to resend, in seconds. The default time is 10 seconds.

persend *number*;
The number of packets per send. The default is 1.

phase1 *number timeunit*;
The maximum time it should take to complete phase 1. The default time is 15 seconds.

phase2 *number timeunit*;
The maximum time it should take to complete phase 2. The default time is 10 seconds.

natt_keepalive *number timeunit*;
The interval between sending NAT-Traversal keep-alive packets. The default time is 20 seconds. Set to 0s to disable keep-alive packets.

Listening Port Specification

listen { *statements* **}**
If no *listen* directive is specified, racoon(8) will listen on all available interface addresses. The following is the list of valid statements:

isakmp *address* [[*port*]];
If this is specified, racoon(8) will only listen on the defined *address*. The default port is 500, which is specified by IANA. You can provide more than one address definition.

isakmp_natt *address* [*port*];
Same as **isakmp** but also sets the socket options to accept UDP-encapsulated ESP traffic for NAT-Traversal. If you plan to use NAT-T, you should provide at least one address with port 4500, which is specified by IANA. There is no default.

strict_address;
Requires that all addresses for ISAKMP be bound. This statement will be ignored if you do not specify address definitions.

When running in privilege separation mode, you need to restart racoon(8) to have changes to the *listen* section taken into account.

The *listen* section can also be used to specify the admin socket mode and ownership if racoon was built with support for admin port.

adminsock *path* [*owner group mode*];
The *path*, *owner*, and *group* values specify the socket path, owner, and group. They must be quoted. The defaults are /var/racoon/racoon.sock, UID 0, and GID 0. *mode* is the access mode in octal. The default is 0600.

adminsock disabled;
This directive tells racoon to not listen on the admin socket.

Miscellaneous Global Parameters

gss_id_enc *enctype*;
Older versions of racoon(8) used ISO-Latin-1 as the encoding of the GSS-API identifier attribute. For interoperability with Microsoft Windows' GSS-API authentication scheme, the default encoding has been changed to UTF-16LE. The **gss_id_enc** parameter allows racoon(8) to be configured to use the old encoding for compatibility with existing racoon(8) installations. The following are valid values for *enctype*:

utf-16le
Use UTF-16LE to encode the GSS-API identifier attribute. This is the default encoding. This encoding is compatible with Microsoft Windows.

latin1
Use ISO-Latin-1 to encode the GSS-API identifier attribute. This is the encoding used by older versions of racoon(8).

Remote Nodes Specifications

remote (*address* | **anonymous**) [[*port*]] [**inherit** *parent*] **{** *statements* **}**
Specifies the IKE phase 1 parameters for each remote node. The default port is 500. If **anonymous** is specified, the statements will apply to any peer that does not match a more specific **remote** directive.

Sections with **inherit** *parent* statements (where *parent* is either *address* or a keyword **anonymous**) that have all values predefined to those of a given *parent*. In these sections it is enough to redefine only the changed parameters.

The following are valid statements.

exchange_mode (**main** | **aggressive** | **base**);
Defines the exchange mode for phase 1 when racoon is the initiator. It also means the acceptable exchange mode when racoon is the responder. More than one mode can be specified by separating them with a comma. All of the modes are acceptable. The first exchange mode is what racoon uses when it is the initiator.

doi ipsec_doi;
Means to use IPsec DOI as specified in RFC 2407. You can omit this statement.

situation identity_only;
Means to use SIT_IDENTITY_ONLY as specified in RFC 2407. You can omit this statement.

identifier *idtype*;
This statement is obsolete. Instead, use **my_identifier**.

my_identifier [*qualifier*] *idtype ...*;
Specifies the identifier sent to the remote host and the type to use in the phase 1 negotiation. **address, fqdn**, **user_fqdn**, **keyid**, and **asn1dn** can be used as an *idtype*. The *qualifier* is currently only used for **keyid**, and can be either **file** or **tag**. The possible values are :

my_identifier address [*address*];
The type is the IP address. This is the default type if you do not specify an identifier to use.

my_identifier user_fqdn *string*;
The type is a USER_FQDN (user fully-qualified domain name).

my_identifier fqdn *string*;
The type is a FQDN (fully-qualified domain name).

my_identifier keyid [**file**] *file*;
The type is a KEY_ID, read from the file.

my_identifier keyid tag *string*;
The type is a KEY_ID, specified in the quoted string.

my_identifier asn1dn [*string*];
The type is an ASN.1 distinguished name. If *string* is omitted, racoon(8) will get the DN from the Subject field in the certificate.

xauth_login [*string*];
Specifies the login to use in client-side Hybrid authentication. It is available only if racoon(8) has been built with this option. The associated password is looked up in the pre-shared key files, using the login **string** as the key id.

peers_identifier *idtype ...*;
Specifies the peer's identifier to be received. If it is not defined then racoon(8) will not verify the peer's identifier in ID payload transmitted from the peer. If it is defined, the behavior of the verification depends on the flag of **verify_identifier**. The usage of *idtype* is the same as **my_identifier** except that the individual component values of an **asn1dn** identifier may specified as * to match any value (e.g. "C=XX, O=MyOrg, OU=*, CN=Mine"). Alternative acceptable peer identifiers may be specified by repeating the **peers_identifier** statement.

verify_identifier (on | **off)**;
If you want to verify the peer's identifier, set this to on. In this case, if the value defined by **peers_identifier** is not the same as the peer's identifier in the ID payload, the negotiation will fail. The default is off.

certificate_type *certspec*;
Specifies a certificate specification. *certspec* is one of followings:

x509 *certfile privkeyfile*;
certfile means a file name of a certificate. *privkeyfile* means a file name of a secret key.

plain_rsa *privkeyfile*;
privkeyfile means a file name of a private key generated by plainrsa-gen(8). Required for RSA authentication.

ca_type *cacertspec*;
Specifies a root certificate authority specification. *cacertspec* is one of followings:
x509 *cacertfile*;
cacertfile means a file name of the root certificate authority. Default is /etc/openssl/cert.pem
mode_cfg (on | **off)**;
Gather network information through ISAKMP mode configuration. Default is off.
weak_phase1_check (on | **off)**;
Tells racoon to act on unencrypted deletion messages during phase 1. This is a small security risk, so the default is off, meaning that racoon will keep on trying to establish a connection even if the user credentials are wrong, for instance.
peers_certfile (**dnssec** | *certfile* | **plain_rsa** *pubkeyfile*);
If **dnssec** is defined, racoon(8) will ignore the CERT payload from the peer, and try to get the peer's certificate from DNS instead. If *certfile* is defined, racoon(8) will ignore the CERT payload from the peer, and will use this certificate as the peer's certificate. If **plain_rsa** is defined, racoon(8) will expect *pubkeyfile* to be the peer's public key that was generated by plainrsa-gen(8).
script *script* **phase1_up**
script *script* **phase1_down**
Shell scripts that get executed when a phase 1 SA goes up or down. Both scripts get either **phase1_up** or **phase1_down** as first argument, and the following variables are set in their environment:
LOCAL_ADDR
The local address of the phase 1 SA.
LOCAL_PORT
The local port used for IKE for the phase 1 SA.
REMOTE_ADDR
The remote address of the phase 1 SA.
REMOTE_PORT
The remote port used for IKE for the phase 1 SA.
The following variables are only set if **mode_cfg** was enabled:
INTERNAL_ADDR4
An IPv4 internal address obtained by ISAKMP mode config.
INTERNAL_NETMASK4
An IPv4 internal netmask obtained by ISAKMP mode config.
INTERNAL_CIDR4
An IPv4 internal netmask obtained by ISAKMP mode config, in CIDR notation.
INTERNAL_DNS4
The first internal DNS server IPv4 address obtained by ISAKMP mode config.
INTERNAL_DNS4_LIST
A list of internal DNS servers IPv4 address obtained by ISAKMP mode config, separated by spaces.
INTERNAL_WINS4
The first internal WINS server IPv4 address obtained by ISAKMP mode config.
INTERNAL_WINS4_LIST
A list of internal WINS servers IPv4 address obtained by ISAKMP mode config, separated by spaces.
SPLIT_INCLUDE
The space separated list of IPv4 addresses and masks (address slash mask) that define the networks to be encrypted (as opposed to the default where all the traffic should be encrypted) ; obtained by ISAKMP mode config ; SPLIT_INCLUDE and SPLIT_LOCAL are mutually exclusive.

SPLIT_LOCAL
: The space separated list of IPv4 addresses and masks (address slash mask) that define the networks to be considered local, and thus excluded from the tunnels ; obtained by ISAKMP mode config.

SPLIT_INCLUDE_CIDR
: Same as SPLIT_INCLUDE, with netmasks in CIDR notation.

SPLIT_LOCAL_CIDR
: Same as SPLIT_LOCAL, with netmasks in CIDR notation.

DEFAULT_DOMAIN
: The DNS default domain name obtained by ISAKMP mode config.

send_cert (on | **off)**;
: If you do not want to send a certificate, set this to off. The default is on.

send_cr (on | **off)**;
: If you do not want to send a certificate request, set this to off. The default is on.

verify_cert (on | **off)**;
: By default, the identifier sent by the remote host (as specified in its **my_identifier** statement) is compared with the credentials in the certificate used to authenticate the remote host as follows:

Type **asn1dn:**
: The entire certificate subject name is compared with the identifier, e.g. "C=XX, O=YY, ...".

Type **address, fqdn, or user_fqdn:**
: The certificate's subjectAltName is compared with the identifier.

If the two do not match the negotiation will fail. If you do not want to verify the identifier using the peer's certificate, set this to off.

lifetime time *number timeunit*;
: Define a lifetime of a certain time which will be proposed in the phase 1 negotiations. Any proposal will be accepted, and the attribute(s) will not be proposed to the peer if you do not specify it (them). They can be individually specified in each proposal.

ike_frag (on | **off** | **force)**;
: Enable receiver-side IKE fragmentation if racoon(8) has been built with this feature. If set to on, racoon will advertise itself as being capable of receiving packets split by IKE fragmentation. This extension is there to work around broken firewalls that do not work with fragmented UDP packets. IKE fragmentation is always enabled on the sender-side, and it is used if the peer advertises itself as IKE fragmentation capable. By selecting force, IKE Fragmentation will be used when racoon is acting as the initiator even before the remote peer has advertised itself as IKE fragmentation capable.

esp_frag *fraglen*;
: This option is only relevant if you use NAT traversal in tunnel mode. Its purpose is to work around broken DSL routers that reject UDP fragments, by fragmenting the IP packets before ESP encapsulation. The result is ESP over UDP of fragmented packets instead of fragmented ESP over UDP packets (i.e., IP:UDP:ESP:frag(IP) instead of frag(IP:UDP:ESP:IP)). *fraglen* is the maximum size of the fragments. 552 should work anywhere, but the higher *fraglen* is, the better the performance.

Note that because PMTU discovery is broken on many sites, you will have to use MSS clamping if you want TCP to work correctly.

initial_contact (on | **off)**;
: Enable this to send an INITIAL-CONTACT message. The default value is **on**. This message is useful only when the responder implementation chooses an old SA when there are multiple SAs with different established time and the initiator reboots. If racoon did not send the message, the responder would use an old SA even when a new SA was established.

For systems that use a KAME derived IPSEC stack, the sysctl(8) variable net.key.preferred_oldsa can be used to control this preference. When the value is zero, the stack always uses a new SA.

passive (on | **off)**;
: If you do not want to initiate the negotiation, set this to on. The default value is **off**. It is useful for a server.

proposal_check *level*;
: Specifies the action of lifetime length, key length, and PFS of the phase 2 selection on the responder side, and the action of lifetime check in phase 1. The default level is **strict**. If the *level* is:

  obey The responder will obey the initiator anytime.

  strict
  : If the responder's lifetime length is longer than the initiator's or the responder's key length is shorter than the initiator's, the responder will use the initiator's value. Otherwise, the proposal will be rejected. If PFS is not required by the responder, the responder will obey the proposal. If PFS is required by both sides and the responder's group is not equal to the initiator's, then the responder will reject the proposal.

  claim If the responder's lifetime length is longer than the initiator's or the responder's key length is shorter than the initiator's, the responder will use the initiator's value. If the responder's lifetime length is shorter than the initiator's, the responder uses its own length AND sends a RESPONDER-LIFETIME notify message to an initiator in the case of lifetime (phase 2 only). For PFS, this directive behaves the same as **strict**.

  exact If the initiator's lifetime or key length is not equal to the responder's, the responder will reject the proposal. If PFS is required by both sides and the responder's group is not equal to the initiator's, then the responder will reject the proposal.

support_proxy (on | **off)**;
: If this value is set to on, then both values of ID payloads in the phase 2 exchange are always used as the addresses of end-point of IPsec-SAs. The default is off.

generate_policy (on | **off** | **require** | **unique)**;
: This directive is for the responder. Therefore you should set **passive** to on in order that racoon(8) only becomes a responder. If the responder does not have any policy in SPD during phase 2 negotiation, and the directive is set to on, then racoon(8) will choose the first proposal in the SA payload from the initiator, and generate policy entries from the proposal. It is useful to negotiate with clients whose IP address is allocated dynamically. Note that an inappropriate policy might be installed into the responder's SPD by the initiator, so other communications might fail if such policies are installed due to a policy mismatch between the initiator and the responder. **on** and **require** values mean the same thing (generate a require policy). **unique** tells racoon to set up unique policies, with a monotoning increasing reqid number (between 1 and IPSEC_MANUAL_REQID_MAX). This directive is ignored in the initiator case. The default value is **off**.

nat_traversal (on | **off** | **force)**;
: This directive enables use of the NAT-Traversal IPsec extension (NAT-T). NAT-T allows one or both peers to reside behind a NAT gateway (i.e., doing address- or port-translation). If a NAT gateway is detected during the phase 1 handshake, racoon will attempt to negotiate the use of NAT-T with the remote peer. If the negotiation succeeds, all ESP and AH packets for the given connection will be encapsulated into UDP datagrams (port 4500, by default). Possible values are:

on NAT-T is used when a NAT gateway is detected between the peers.
off NAT-T is not proposed/accepted. This is the default.
force NAT-T is used regardless of whether a NAT gateway is detected between the peers or not.

Please note that NAT-T support is a compile-time option. Although it is enabled in the source distribution by default, it may not be available in your particular build. In that case you will get a warning when using any NAT-T related config options.

dpd_delay *delay*;

This option activates the DPD and sets the time (in seconds) allowed between 2 proof of liveliness requests. The default value is **0**, which disables DPD monitoring, but still negotiates DPD support.

dpd_retry *delay*;

If **dpd_delay** is set, this sets the delay (in seconds) to wait for a proof of liveliness before considering it as failed and send another request. The default value is **5**.

dpd_maxfail *number*;

If **dpd_delay** is set, this sets the maximum number of liveliness proofs to request (without reply) before considering the peer is dead. The default value is **5**.

nonce_size *number*;

define the byte size of nonce value. Racoon can send any value although RFC2409 specifies that the value MUST be between 8 and 256 bytes. The default size is 16 bytes.

ph1id *number*;

An optional number to identify the remote proposal and to link it only with sainfos who have the same number. Defaults to 0.

proposal { *sub-substatements* }

encryption_algorithm *algorithm*;

Specifies the encryption algorithm used for the phase 1 negotiation. This directive must be defined. *algorithm* is one of following: **des, 3des, blowfish, cast128, aes, camellia** for Oakley. For other transforms, this statement should not be used.

hash_algorithm *algorithm*;

Defines the hash algorithm used for the phase 1 negotiation. This directive must be defined. *algorithm* is one of following: **md5, sha1, sha256, sha384, sha512** for Oakley.

authentication_method *type*;

Defines the authentication method used for the phase 1 negotiation. This directive must be defined. *type* is one of: **pre_shared_key**, **rsasig** (for plain RSA authentication), **gssapi_krb**, **hybrid_rsa_server**, **hybrid_rsa_client**, **xauth_rsa_server**, **xauth_rsa_client**, **xauth_psk_server** or **xauth_psk_client**.

dh_group *group*;

Defines the group used for the Diffie-Hellman exponentiations. This directive must be defined. *group* is one of following: **modp768**, **modp1024**, **modp1536**, **modp2048**, **modp3072**, **modp4096**, **modp6144**, **modp8192**. Or you can define 1, 2, 5, 14, 15, 16, 17, or 18 as the DH group number. When you want to use aggressive mode, you must define the same DH group in each proposal.

lifetime time *number timeunit*;

Defines the lifetime of the phase 1 SA proposal. Refer to the description of the **lifetime** directive defined in the **remote** directive.

gss_id *string*;
Defines the GSS-API endpoint name, to be included as an attribute in the SA, if the **gssapi_krb** authentication method is used. If this is not defined, the default value of `host/hostname` is used, where hostname is the value returned by the `hostname`(1) command.

Policy Specifications

The policy directive is obsolete, policies are now in the SPD. `racoon`(8) will obey the policy configured into the kernel by `setkey`(8), and will construct phase 2 proposals by combining **sainfo** specifications in **racoon.conf**, and policies in the kernel.

Sainfo Specifications

sainfo (*local_id* | **anonymous**) (*remote_id* | **clientaddr** | **anonymous**) [**from** *idtype* [*string*]] [**group** *string*] **{** *statements* **}**
Defines the parameters of the IKE phase 2 (IPsec-SA establishment).

The *local_id* and *remote_id* strings are constructed like:

address *address* [*/ prefix*] [[*port*]] *ul_proto*

or

subnet *address* [*/ prefix*] [[*port*]] *ul_proto*

An id string should be expressed to match the exact value of an ID payload. This is not like a filter rule. For example, if you define 3ffe:501:4819::/48 as *local_id*. 3ffe:501:4819:1000:/64 will not match. In the case of a longest prefix (selecting a single host), *address* instructs to send ID type of ADDRESS while *subnet* instructs to send ID type of SUBNET. Otherwise, these instructions are identical.

The **anonymous** keyword can be used to match any id. The **clientaddr** keyword can be used to match a remote id that is equal to either the peer ip address or the mode_cfg ip address (if assigned). This can be useful to restrict policy generation when racoon is acting as a client gateway for peers with dynamic ip addresses.

The **from** keyword allows an sainfo to only match for peers that use a specific phase1 id value during authentication. The **group** keyword allows an XAuth group membership check to be performed for this sainfo section. When the mode_cfg auth source is set to **system** or **ldap**, the XAuth user is verified to be a member of the specified group before allowing a matching SA to be negotiated.

pfs_group *group*;
define the group of Diffie-Hellman exponentiations. If you do not require PFS then you can omit this directive. Any proposal will be accepted if you do not specify one. *group* is one of following: **modp768**, **modp1024**, **modp1536**, **modp2048**, **modp3072**, **modp4096**, **modp6144**, **modp8192**. Or you can define 1, 2, 5, 14, 15, 16, 17, or 18 as the DH group number.

lifetime time *number timeunit*;
define how long an IPsec-SA will be used, in timeunits. Any proposal will be accepted, and no attribute(s) will be proposed to the peer if you do not specify it(them). See the **proposal_check** directive.

remoteid *number*;
Sainfos will only be used if their remoteid matches the ph1id of the remote section used for phase 1. Defaults to 0, which is also the default for ph1id.

my_identifier *idtype ...*;
is obsolete. It does not make sense to specify an identifier in the phase 2.

racoon(8) does not have a list of security protocols to be negotiated. The list of security protocols are passed by SPD in the kernel. Therefore you have to define all of the potential algorithms in the phase 2 proposals even if there are algorithms which will not be used. These algorithms are define by using the following three directives, with a single comma as the separator. For algorithms that can take variable-length keys, algorithm names can be followed by a key length, like "blowfish 448". racoon(8) will compute the actual phase 2 proposals by computing the permutation of the specified algorithms, and then combining them with the security protocol specified by the SPD. For example, if **des**, **3des**, **hmac_md5**, and **hmac_sha1** are specified as algorithms, we have four combinations for use with ESP, and two for AH. Then, based on the SPD settings, racoon(8) will construct the actual proposals. If the SPD entry asks for ESP only, there will be 4 proposals. If it asks for both AH and ESP, there will be 8 proposals. Note that the kernel may not support the algorithm you have specified.

encryption_algorithm *algorithms*;
des, **3des**, **des_iv64**, **des_iv32**, **rc5**, **rc4**, **idea**, **3idea**, **cast128**, **blowfish**, **null_enc**, **twofish**, **rijndael**, **aes**, **camellia** (used with ESP)

authentication_algorithm *algorithms*;
des, **3des**, **des_iv64**, **des_iv32**, **hmac_md5**, **hmac_sha1**, **hmac_sha256**, **hmac_sha384**, **hmac_sha512**, **non_auth** (used with ESP authentication and AH)

compression_algorithm *algorithms*;
deflate (used with IPComp)

Logging level

log *level*;
Defines the logging level. *level* is one of following: **error**, **warning**, **notify**, **info**, **debug** and **debug2**. The default is **info**. If you set the logging level too high on slower machines, IKE negotiation can fail due to timing constraint changes.

Specifies the way to pad

padding { *statements* }
specifies the padding format. The following are valid statements:

randomize (on | off);
Enables the use of a randomized value for padding. The default is on.

randomize_length (on | off);
The pad length will be random. The default is off.

maximum_length *number*;
Defines a maximum padding length. If **randomize_length** is off, this is ignored. The default is 20 bytes.

exclusive_tail (on | off);
Means to put the number of pad bytes minus one into the last part of the padding. The default is on.

strict_check (on | off);
Means to constrain the peer to set the number of pad bytes. The default is off.

ISAKMP mode configuration settings

mode_cfg { *statements* }
Defines the information to return for remote hosts' ISAKMP mode config requests. Also defines the authentication source for remote peers authenticating through Xauth.

The following are valid statements:

auth_source (system | **radius** | **pam** | **ldap)**;
: Specifies the source for authentication of users through Xauth. *system* means to use the Unix user database. This is the default. *radius* means to use a RADIUS server. It works only if racoon(8) was built with libradius support. Radius configuration is handled by radius.conf(5). *pam* means to use PAM. It works only if racoon(8) was built with libpam support. *ldap* means to use LDAP. It works only if racoon(8) was built with libldap support. LDAP configuration is handled by statements in the **ldapcfg** section.

auth_groups *group1, ...*;
: Specifies the group memberships for Xauth in quoted group name strings. When defined, the authenticating user must be a member of at least one group for Xauth to succeed.

group_source (system | **ldap)**;
: Specifies the source for group validation of users through Xauth. *system* means to use the Unix user database. This is the default. *ldap* means to use LDAP. It works only if racoon(8) was built with libldap support and requires LDAP authentication. LDAP configuration is handled by statements in the **ldapcfg** section.

conf_source (local | **radius** | **ldap)**;
: Specifies the source for IP addresses and netmask allocated through ISAKMP mode config. *local* means to use the local IP pool defined by the **network4** and **pool_size** statements. This is the default. *radius* means to use a RADIUS server. It works only if racoon(8) was built with libradius support and requires RADIUS authentication. RADIUS configuration is handled by radius.conf(5). *ldap* means to use an LDAP server. It works only if racoon(8) was built with libldap support and requires LDAP authentication. LDAP configuration is handled by statements in the **ldapcfg** section.

accounting (none | **system** | **radius** | **pam)**;
: Enables or disables accounting for Xauth logins and logouts. The default is *none* which disable accounting. Specifying *system* enables system accounting through utmp(5). Specifying *radius* enables RADIUS accounting. It works only if racoon(8) was built with libradius support and requires RADIUS authentication. RADIUS configuration is handled by radius.conf(5). Specifying *pam* enables PAM accounting. It works only if racoon(8) was build with libpam support and requires PAM authentication.

pool_size *size*
: Specify the size of the IP address pool, either local or allocated through RADIUS. **conf_source** selects the local pool or the RADIUS configuration, but in both configurations, you cannot have more than *size* users connected at the same time. The default is 255.

network4 *address*;
netmask4 *address*;
: The local IP pool base address and network mask from which dynamically allocated IPv4 addresses should be taken. This is used if **conf_source** is set to *local* or if the RADIUS server returned *255.255.255.254*. Default is *0.0.0.0/0.0.0.0*.

dns4 *addresses*;
: A list of IPv4 addresses for DNS servers, separated by commas, or on multiple **dns4** lines.

wins4 *addresses*;
: A list of IPv4 address for WINS servers. The keyword
nbns4 can also be used as an alias for
wins4.

split_network (include | **local_lan)** *network/mask, ...*
: The network configuration to send, in CIDR notation (e.g. 192.168.1.0/24). If **include** is specified, the tunnel should be only used to encrypt the indicated destinations ; otherwise, if **local_lan** is used, everything will pass through the tunnel but those destinations.

default_domain *domain*;
The default DNS domain to send.
split_dns *domain, ...*
The split dns configuration to send, in quoted domain name strings. This list can be used to describe a list of domain names for which a peer should query a modecfg assigned dns server. DNS queries for all other domains would be handled locally. (Cisco VPN client only).
banner *path*;
The path of a file displayed on the client at connection time. Default is */etc/motd*.
auth_throttle *delay*;
On each failed Xauth authentication attempt, refuse new attempts for a set *delay* of seconds. This is to avoid dictionary attacks on Xauth passwords. Default is one second. Set to zero to disable authentication delay.
pfs_group *group*;
Sets the PFS group used in the client proposal (Cisco VPN client only). Default is 0.
save_passwd (on | **off)**;
Allow the client to save the Xauth password (Cisco VPN client only). Default is off.

Ldap configuration settings

ldapcfg { *statements* }
Defines the parameters that will be used to communicate with an ldap server for **xauth** authentication.

The following are valid statements:
version (2 | **3)**;
The ldap protocol version used to communicate with the server. The default is **3**.
host *(hostname* | *address)*;
The host name or ip address of the ldap server. The default is **localhost**.
port *number;*
The port that the ldap server is configured to listen on. The default is **389**.
base *distinguished name;*
The ldap search base. This option has no default value.
subtree (on | **off)**;
Use the subtree ldap search scope. Otherwise, use the one level search scope. The default is **off**.
bind_dn *distinguished name;*
The user dn used to optionally bind as before performing ldap search operations. If this option is not specified, anonymous binds are used.
bind_pw *string;*
The password used when binding as **bind_dn**.
attr_user *attribute name;*
The attribute used to specify a users name in an ldap directory. For example, if a user dn is "cn=jdoe,dc=my,dc=net" then the attribute would be "cn". The default value is **cn**.
attr_addr *attribute name;*
attr_mask *attribute name;*
The attributes used to specify a users network address and subnet mask in an ldap directory. These values are forwarded during mode_cfg negotiation when the conf_source is set to ldap. The default values are **racoon-address** and **racoon-netmask**.
attr_group *attribute name;*
The attribute used to specify a group name in an ldap directory. For example, if a group dn is "cn=users,dc=my,dc=net" then the attribute would be "cn". The default value is **cn**.

attr_member *attribute name;*
The attribute used to specify group membership in an ldap directory. The default value is **member**.

Special directives

complex_bundle (on | off);
defines the interpretation of proposal in the case of SA bundle. Normally "IP AH ESP IP payload" is proposed as "AH tunnel and ESP tunnel". The interpretation is more common to other IKE implementations, however, it allows very limited set of combinations for proposals. With the option enabled, it will be proposed as "AH transport and ESP tunnel". The default value is **off**.

Pre-shared key File

The pre-shared key file defines pairs of identifiers and corresponding shared secret keys which are used in the pre-shared key authentication method in phase 1. The pair in each line is separated by some number of blanks and/or tab characters like in the `hosts`(5) file. Key can include blanks because everything after the first blanks is interpreted as the secret key. Lines starting with '#' are ignored. Keys which start with '`0x`' are interpreted as hexadecimal strings. Note that the file must be owned by the user ID running `racoon`(8) (usually the privileged user), and must not be accessible by others.

EXAMPLES

The following shows how the remote directive should be configured.

```
path pre_shared_key "/usr/local/v6/etc/psk.txt" ;
remote anonymous
{
        exchange_mode aggressive,main,base;
        lifetime time 24 hour;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method pre_shared_key;
                dh_group 2;
        }
}

sainfo anonymous
{
        pfs_group 2;
        lifetime time 12 hour ;
        encryption_algorithm 3des, blowfish 448, twofish, rijndael ;
        authentication_algorithm hmac_sha1, hmac_md5 ;
        compression_algorithm deflate ;
}
```

If you are configuring plain RSA authentication, the remote directive should look like the following:

```
path certificate "/usr/local/v6/etc" ;
remote anonymous
{
        exchange_mode main,base ;
        lifetime time 12 hour ;
        certificate_type plain_rsa "/usr/local/v6/etc/myrsakey.priv";
        peers_certfile plain_rsa "/usr/local/v6/etc/yourrsakey.pub";
```

```
        proposal {
                        encryption_algorithm aes ;
                        hash_algorithm sha1 ;
                        authentication_method rsasig ;
                        dh_group 2 ;
        }
}
```

The following is a sample for the pre-shared key file.

```
10.160.94.3     mekmitasdigoat
172.16.1.133    0x12345678
194.100.55.1    whatcertificatereally
3ffe:501:410:ffff:200:86ff:fe05:80fa    mekmitasdigoat
3ffe:501:410:ffff:210:4bff:fea2:8baa    mekmitasdigoat
foo@kame.net    mekmitasdigoat
foo.kame.net    hoge
```

SEE ALSO

racoon(8), racoonctl(8), setkey(8)

HISTORY

The **racoon.conf** configuration file first appeared in the “YIPS” Yokogawa IPsec implementation.

BUGS

Some statements may not be handled by racoon(8) yet.

Diffie-Hellman computation can take a very long time, and may cause unwanted timeouts, specifically when a large D-H group is used.

SECURITY CONSIDERATIONS

The use of IKE phase 1 aggressive mode is not recommended, as described in http://www.kb.cert.org/vuls/id/886601.

NAME

radius.conf — RADIUS client configuration file

SYNOPSIS

/etc/radius.conf

DESCRIPTION

radius.conf contains the information necessary to configure the RADIUS client library. It is parsed by rad_config(3). The file contains one or more lines of text, each describing a single RADIUS server which will be used by the library. Leading white space is ignored, as are empty lines and lines containing only comments.

A RADIUS server is described by three to five fields on a line:

Service type
Server host
Shared secret
Timeout
Retries

The fields are separated by white space. The '#' character at the beginning of a field begins a comment, which extends to the end of the line. A field may be enclosed in double quotes, in which case it may contain white space and/or begin with the '#' character. Within a quoted string, the double quote character can be represented by '\"', and the backslash can be represented by '\\'. No other escape sequences are supported.

The first field gives the service type, either auth for RADIUS authentication or acct for RADIUS accounting. If a single server provides both services, two lines are required in the file. Earlier versions of this file did not include a service type. For backward compatibility, if the first field is not auth or acct the library behaves as if auth were specified, and interprets the fields in the line as if they were fields two through five.

The second field specifies the server host, either as a fully qualified domain name or as a dotted-quad IP address. The host may optionally be followed by a ':' and a numeric port number, without intervening white space. If the port specification is omitted, it defaults to the radius or radacct service in the /etc/services file for service types auth and acct, respectively. If no such entry is present, the standard ports 1812 and 1813 are used.

The third field contains the shared secret, which should be known only to the client and server hosts. It is an arbitrary string of characters, though it must be enclosed in double quotes if it contains white space. The shared secret may be any length, but the RADIUS protocol uses only the first 128 characters. N.B., some popular RADIUS servers have bugs which prevent them from working properly with secrets longer than 16 characters.

The fourth field contains a decimal integer specifying the timeout in seconds for receiving a valid reply from the server. If this field is omitted, it defaults to 3 seconds.

The fifth field contains a decimal integer specifying the maximum number of attempts that will be made to authenticate with the server before giving up. If omitted, it defaults to 3 attempts. Note, this is the total number of attempts and not the number of retries.

Up to 10 RADIUS servers may be specified for each service type. The servers are tried in round-robin fashion, until a valid response is received or the maximum number of tries has been reached for all servers.

The standard location for this file is /etc/radius.conf. But an alternate pathname may be specified in the call to rad_config(3). Since the file contains sensitive information in the form of the shared secrets, it should not be readable except by root.

FILES

`/etc/radius.conf`

EXAMPLES

```
# A simple entry using all the defaults:
acct  radius1.domain.com  OurLittleSecret

# A server still using the obsolete RADIUS port, with increased
# timeout and maximum tries:
auth  auth.domain.com:1645  "I can't see you"  5  4

# A server specified by its IP address:
auth  192.168.27.81  $X*#..38947ax-+=
```

SEE ALSO

`libradius`(3)

C. Rigney, et al, *Remote Authentication Dial In User Service (RADIUS)*, RFC 2138.

C. Rigney, *RADIUS Accounting*, RFC 2139.

AUTHORS

This documentation was written by John Polstra, and donated to the FreeBSD project by Juniper Networks, Inc.

NAME

ranlib — a.out archive (library) table-of-contents format

SYNOPSIS

```
#include <ranlib.h>
```

DESCRIPTION

The archive table-of-contents command **ranlib** creates a table of contents for archives, containing object files, to be used by the link-editor ld(1). It operates on archives created with the utility ar(1).

The **ranlib** function prepends a new file to the archive which has three separate parts. The first part is a standard archive header, which has a special name field, "__.SYMDEF".

The second part is a “long” followed by a list of ranlib structures. The long is the size, in bytes, of the list of ranlib structures. Each of the ranlib structures consists of a zero based offset into the next section (a string table of symbols) and an offset from the beginning of the archive to the start of the archive file which defines the symbol. The actual number of ranlib structures is this number divided by the size of an individual ranlib structure.

The third part is a “long” followed by a string table. The long is the size, in bytes of the string table.

SEE ALSO

ar(1), ranlib(1)

BUGS

The <ranlib.h> header file, and the **ranlib** manual page, do not describe the table-of-contents used by ELF systems, which is that from the AT&T System V.4 UNIX ABI.

NAME

rc.conf — system startup configuration file

DESCRIPTION

The **rc.conf** file specifies which services are enabled during system startup by the startup scripts invoked by /etc/rc (see rc(8)), and the shutdown scripts invoked by /etc/rc.shutdown. The **rc.conf** file is a shell script that is sourced by rc(8), meaning that **rc.conf** must contain valid shell commands.

Listed below are the standard **rc.conf** variables that may be set, the values to which each may be set, a brief description of what each variable does, and a reference to relevant manual pages. Third party packages may test for additional variables.

Most variables are one of two types: enabling variables or flags variables. Enabling variables, such as **inetd**, are generally named after the program or the system they enable, and are set to 'YES' or 'NO'. Flags variables, such as **inetd_flags** have the same name with "_flags" appended, and determine what arguments are passed to the program if it is enabled.

If a variable that rc(8) expects to be set is not set, or the value is not one of the allowed values, a warning will be printed.

By default, **rc.conf** reads /etc/defaults/rc.conf (if it is readable) to obtain default values for various variables, and the end-user may override these by appending appropriate entries to the end of **rc.conf**.

rc.d(8) scripts that use **load_rc_config** from rc.subr(8) also support sourcing an optional end-user provided per-script override file /etc/rc.conf.d/*service*, (where *service* is the contents of the **name** variable in the rc.d(8) script). This may contain variable overrides, including allowing the end-user to override various **run_rc_command** rc.d(8) control variables, and thus changing the operation of the script without requiring editing of the script.

Overall control

do_rcshutdown 'YES' or 'NO'. If set to 'NO', shutdown(8) will not run /etc/rc.shutdown.

rcshutdown_rcorder_flags
A string. Extra arguments to the rcorder(8) run by /etc/rc.shutdown.

rcshutdown_timeout
A number. If non-blank, use this as the number of seconds to run a watchdog timer for which will terminate /etc/rc.shutdown if the timer expires before the shutdown script completes.

rc_configured 'YES' or 'NO'. If not set to 'YES' then the system will drop into single-user mode during boot.

rc_fast_and_loose If set to a non-empty string, each script in /etc/rc.d will be executed in the current shell rather than a sub shell. This may be faster on slow machines that have an expensive fork(2) operation.

Note: Use this at your own risk! A rogue command or script may inadvertently prevent boot to multiuser.

rc_rcorder_flags A string. Extra arguments to the rcorder(8) run by /etc/rc.

Basic network configuration

defaultroute A string. Default IPv4 network route. If empty or not set, then the contents of /etc/mygate (if it exists) are used.

defaultroute6 A string. Default IPv6 network route. If empty or not set, then the contents of `/etc/mygate6` (if it exists) are used.

domainname A string. NIS (YP) domain of host. If empty or not set, then the contents of `/etc/defaultdomain` (if it exists) are used.

force_down_interfaces
A space separated list of interface names. These interfaces will be configured down when going from multiuser to singleuser mode or on system shutdown.

This is important for some stateful interfaces, for example PPP over ISDN connections that cost money by connection time or PPPoE interfaces which have no direct means of noticing "disconnect" events.

All active `pppoe`(4) and `ippp`(4) interfaces will be automatically added to this list.

hostname A string. Name of host. If empty or not set, then the contents of `/etc/myname` (if it exists) are used.

Boottime file-system and swap configuration

critical_filesystems_local
A string. File systems mounted very early in the system boot before networking services are available. Usually `/var` is part of this, because it is needed by services such as `dhclient`(8) which may be required to get the network operational.

critical_filesystems_remote
A string. File systems such as `/usr` that may require network services to be available to mount, that must be available early in the system boot for general services to use.

fsck_flags A string. A file system is checked with `fsck`(8) during boot before mounting it. This option may be used to override the default command-line options passed to the `fsck`(8) program.

When set to **`-y`**, `fsck`(8) assumes yes as the answer to all operator questions during file system checks. This might be important with hosts where the administrator does not have access to the console and an unsuccessful shutdown must not make the host unbootable even if the file system checks would fail in preen mode.

no_swap 'YES' or 'NO'. Set the **no_swap** variable to 'YES' if you have configured your system with no swap on purpose. If not set to 'YES', and no swap devices are configured, the system will warn you.

swapoff 'YES' or 'NO'. Remove block-type swap devices at shutdown time. Useful if swapping onto RAIDframe devices.

One-time actions to perform or programs to run on boot-up

accounting 'YES' or 'NO'. Enables process accounting with `accton`(8). Requires `/var/account/acct` to exist.

clear_tmp 'YES' or 'NO'. Clear /tmp after reboot.

dmesg 'YES' or 'NO'. Create `/var/run/dmesg.boot` from the output of `dmesg`(8). Passes **dmesg_flags**.

lkm 'YES' or 'NO'. Runs `/etc/rc.lkm`.

mixerctl 'YES' or 'NO'. Read `mixerctl.conf`(5) for how to set mixer values. List in **mixerctl_mixers** the devices whose settings are to be saved at shutdown and restored at start-up.

newsyslog 'YES' or 'NO'. Run **newsyslog** to trim logfiles before syslogd starts. Intended for laptop users. Passes **newsyslog_flags**.

per_user_tmp 'YES' or 'NO'. Enables a per-user `/tmp` directory. **per_user_tmp_dir** can be used to override the default location of the "real" temporary directories, "`/private/tmp`".

savecore 'YES' or 'NO'. Runs the `savecore`(8) utility. Passes **savecore_flags**. The directory where crash dumps are stored is specified by **savecore_dir**. The default setting is "`/var/crash`".

tpctl 'YES' or 'NO'. Run `tpctl`(8) to calibrate touch panel device. Passes **tpctl_flags**.

update_motd 'YES' or 'NO'. Updates the NetBSD version string in the `/etc/motd` file to reflect the version of the running kernel. See `motd`(5).

veriexec 'YES' or 'NO'. Load Veriexec fingerprints during startup. Read `veriexecctl`(8) for more information.

virecover 'YES' or 'NO'. Send notification mail to users if any recoverable files exist in `/var/tmp/vi.recover`. Read `virecover`(8) for more information.

System security setting

securelevel A number. The system securelevel is set to the specified value early in the boot process, before any external logins, or other programs that run users job, are started. If set to nothing, the default action is taken, as described in `init`(8), which contains definitive information about the system securelevel. Note that setting **securelevel** to 0 in **rc.conf** will actually result in the system booting with securelevel set to 1, as `init`(8) will raise the level when `rc`(8) completes.

permit_nonalpha Allow passwords to include non-alpha characters, usually to allow NIS/YP netgroups.

veriexec_strict A number. Controls the strict level of Veriexec. Level 0 is learning mode, used when building the signatures file. It will only output messages but will not enforce anything. Level 1 will only prevent access to files with a fingerprint mismatch. Level 2 will also deny writing to and removing of monitored files, as well as enforce access type (as specified in the signatures file). Level 3 will take a step further and prevent access to files that are not monitored.

veriexec_verbose A number. Controls the verbosity of Veriexec. Recommended operation is at level 0, verbose output (mostly used when building the signatures file) is at level 1. Level 2 is for debugging only and should not be used.

veriexec_flags A string. Flags to pass to the **veriexecctl** command.

Networking startup

altqd 'YES' or 'NO'. ALTQ configuration/monitoring daemon. Passes **altqd_flags**.

auto_ifconfig 'YES' or 'NO'. Sets the **net_interfaces** variable (see below) to the output of `ifconfig`(8) with the "`-l`" flag and suppresses warnings about interfaces in this list that do not have an ifconfig file or variable.

dhclient 'YES' or 'NO'. Set to 'YES' to configure some or all network interfaces using the ISC DHCP client. If you set **dhclient** to 'YES', you must either have `/var` in **critical_filesystems_local**, as part of `/`, or direct the DHCP client to store the leases file on the root file system by modifying the **dhclient_flags** variable. You must not provide ifconfig information or ifaliases information for any interface that is to be configured using the DHCP client. Interface aliases can be set up in the DHCP client con-

figuration file if needed - see `dhclient.conf`(5) for details.

Passes **dhclient_flags** to the DHCP client. See `dhclient`(8) for complete documentation. If you wish to configure all broadcast network interfaces using the DHCP client, you can leave this blank. To configure only specific interfaces, name the interfaces to be configured on the command line.

If you must run the DHCP client before mounting critical file systems, then you should specify an alternate location for the DHCP client's lease file in the **dhclient_flags** variable - for example, "-lf /tmp/dhclient.leases".

dhcpcd_flags Additional arguments to pass to `dhcpcd`(8) when requesting configuration via **ifconfig_xxN**.

flushroutes 'YES' or 'NO'. Flushes the route table on networking startup. Useful when coming up to multiuser mode after going down to single-user mode.

hostapd 'YES' or 'NO'. Runs `hostapd`(8), the authenticator for IEEE 802.11 networks.

ifaliases_∗ A string. List of '*address netmask*' pairs to configure additional network addresses for the given configured interface "∗" (e.g. **ifaliases_le0**). If *netmask* is "-", then use the default netmask for the interface.

ifaliases_∗ covers limited cases only and considered unrecommended. We recommend using `/etc/ifconfig.xxN` with multiple lines instead.

ifwatchd 'YES' or 'NO'. Monitor dynamic interfaces and perform actions upon address changes. Passes **ifwatchd_flags**.

ip6mode A string. An IPv6 node can be a router (nodes that forward packet for others) or a host (nodes that do not forward). A host can be autoconfigured based on the information advertised by adjacent IPv6 routers. By setting **ip6mode** to "`router`", "`host`", or "`autohost`", you can configure your node as a router, a non-autoconfigured host, or an autoconfigured host. Invalid values will be ignored, and the node will be configured as a non-autoconfigured host. You may want to check **rtsol** and **rtsold** as well, if you set the variable to "`autohost`".

ip6uniquelocal 'YES' or 'NO'. If **ip6mode** is equal to "`router`" and **ip6uniquelocal** is set to 'NO' a reject route will be installed on boot to avoid misconfiguration relating to unique-local addresses. If set to 'YES' the reject route won't be installed.

ipfilter 'YES' or 'NO'. Runs `ipf`(8) to load in packet filter specifications from `/etc/ipf.conf` at network boot time, before any interfaces are configured. See `ipf.conf`(5).

ipfs 'YES' or 'NO'. Runs `ipfs`(8) to save and restore information for ipnat and ipfilter state tables. The information is stored in `/var/db/ipf/ipstate.ipf` and `/var/db/ipf/ipnat.ipf`. Passes **ipfs_flags**.

ipmon 'YES' or 'NO'. Runs `ipmon`(8) to read `ipf`(8) packet log information and log it to a file or the system log. Passes **ipmon_flags**.

ipmon_flags A string. Specifies arguments to supply to `ipmon`(8). Defaults to "`-ns`". A typical example would be "**`-nD /var/log/ipflog`**" to have `ipmon`(8) log directly to a file bypassing `syslogd`(8). If the "-D" argument is used, remember to modify `/etc/newsyslog.conf` accordingly; for example:

```
/var/log/ipflog  640  10  100  *  Z  /var/run/ipmon.pid
```

ipnat 'YES' or 'NO'. Runs `ipnat`(8) to load in the IP network address translation (NAT) rules from `/etc/ipnat.conf` at network boot time, before any interfaces are configured. See `ipnat.conf`(5).

ipsec 'YES' or 'NO'. Runs `setkey`(8) to load in IPsec manual keys and policies from `/etc/ipsec.conf` at network boot time, before any interfaces are configured.

net_interfaces A string. The list of network interfaces to be configured at boot time. For each interface "xxN", the system first looks for ifconfig parameters in `/etc/ifconfig.xxN` and then in the variable **ifconfig_xxN**. If this variable is equal to "dhcp", `dhcpcd`(8) is started for the interface. Otherwise the contents of the file or the variable are handed to ifconfig after the interface name. If **auto_ifconfig** is set to "NO" and neither the file nor the variable is found, a warning is printed. Refer to `ifconfig.if`(5) for more details on `/etc/ifconfig.xxN`.

ntpdate 'YES' or 'NO'. Runs `ntpdate`(8) to set the system time from one of the hosts in **ntpdate_hosts**. If **ntpdate_hosts** is empty, it will attempt to find a list of hosts in `/etc/ntp.conf`. Passes **ntpdate_flags**.

ppp_peers A string. If **ppp_peers** is not empty, then `/etc/rc.d/ppp` will check each word in **ppp_peers** for a corresponding ppp configuration file in `/etc/ppp/peers` and will call `pppd`(8) with the "call **peer**" option.

racoon 'YES' or 'NO'. Runs `racoon`(8), the IKE (ISAKMP/Oakley) key management daemon.

rtsol 'YES' or 'NO'. Run `rtsol`(8), router solicitation command for IPv6 hosts. On nomadic hosts like notebook computers, you may want to enable **rtsold** as well. Passes **rtsol_flags**. This is only for autoconfigured IPv6 hosts, so set **ip6mode** to "`autohost`" if you use it.

wpa_supplicant 'YES' or 'NO'. Run `wpa_supplicant`(8), WPA/802.11i Supplicant for wireless network devices.

Daemons required by other daemons

inetd 'YES' or 'NO'. Runs the `inetd`(8) daemon to start network server processes (as listed in `/etc/inetd.conf`) as necessary. Passes **inetd_flags**. The "`-l`" flag turns on libwrap connection logging.

rpcbind 'YES' or 'NO'. The `rpcbind`(8) daemon is required for any `rpc`(3) services. These include NFS, NIS, `bootparamd`(8), `rstatd`(8), `rusersd`(8), and `rwalld`(8). Passes **rpcbind_flags**.

Commonly used daemons

cron 'YES' or 'NO'. Run `cron`(8).

lpd 'YES' or 'NO'. Runs `lpd`(8) and passes **lpd_flags**. The "`-l`" flag will turn on extra logging.

named 'YES' or 'NO'. Runs `named`(8) and passes **named_flags**.

named_chrootdir A string. If non-blank and **named** is 'YES', run `named`(8) as the unprivileged user and group 'named', `chroot`(2)ed to **named_chrootdir**. **named_chrootdir**`/var/run/log` will be added to the list of log sockets that `syslogd`(8) listens to.

ntpd 'YES' or 'NO'. Runs `ntpd`(8) and passes **ntpd_flags**.

ntpd_chrootdir A string. If non-blank and **ntpd** is 'YES', run `ntpd`(8) as the unprivileged user and group 'ntpd', `chroot`(2)ed to **ntpd_chrootdir**. **ntpd_chrootdir**`/var/run/log` will be added to the list of log sockets that `syslogd`(8) listens to. This option requires that the kernel has

```
pseudo-device clockctl
```

compiled in, and that `/dev/clockctl` is present.

postfix 'YES' or 'NO'. Starts `postfix`(1) mail system.

sshd 'YES' or 'NO'. Runs `sshd`(8) and passes **sshd_flags**.

syslogd 'YES' or 'NO'. Runs `syslogd`(8) and passes **syslogd_flags**.

timed 'YES' or 'NO'. Runs `timed`(8) and passes **timed_flags**. The "`-M`" option allows `timed`(8) to be a master time source as well as a slave. If you are also running `ntpd`(8), only one machine running both should have the "`-M`" flag given to `timed`(8).

Routing daemons

mrouted 'YES' or 'NO'. Runs `mrouted`(8), the DVMRP multicast routing protocol daemon. Passes **mrouted_flags**.

route6d 'YES' or 'NO'. Runs `route6d`(8), the RIPng routing protocol daemon for IPv6. Passes **route6d_flags**.

routed 'YES' or 'NO'. Runs `routed`(8), the RIP routing protocol daemon. Passes **routed_flags**.

rtsold 'YES' or 'NO'. Runs `rtsold`(8), the IPv6 router solicitation daemon. `rtsold`(8) periodically transmits router solicitation packets to find IPv6 routers on the network. This configuration is mainly for nomadic hosts like notebook computers. Stationary hosts should work fine with just **rtsol**. Passes **rtsold_flags**. This is only for autoconfigured IPv6 hosts, so set **ip6mode** to "`autohost`" if you use it.

Daemons used to boot other hosts over a network

bootparamd 'YES' or 'NO'. Runs `bootparamd`(8), the boot parameter server, with **bootparamd_flags** as options. Used to boot NetBSD and SunOS 4.x systems.

dhcpd 'YES' or 'NO'. Runs `dhcpd`(8), the Dynamic Host Configuration Protocol (DHCP) daemon, for assigning IP addresses to hosts and passing boot information. Passes **dhcpd_flags**.

dhcrelay 'YES' or 'NO'. Runs `dhcrelay`(8). Passes **dhcrelay_flags**.

mopd 'YES' or 'NO'. Runs `mopd`(8), the DEC MOP protocol daemon; used for booting VAX and other DEC machines. Passes **mopd_flags**.

ndbootd 'YES' or 'NO'. Runs `ndbootd`(8), the Sun Network Disk (ND) Protocol server. Passes **ndbootd_flags**.

rarpd 'YES' or 'NO'. Runs `rarpd`(8), the reverse ARP daemon, often used to boot NetBSD and Sun workstations. Passes **rarpd_flags**.

rbootd 'YES' or 'NO'. Runs `rbootd`(8), the HP boot protocol daemon; used for booting HP workstations. Passes **rbootd_flags**.

rtadvd 'YES' or 'NO'. Runs `rtadvd`(8), the IPv6 router advertisement daemon, which is used to advertise information about the subnet to IPv6 end hosts. Passes **rtadvd_flags**. This is only for IPv6 routers, so set **ip6mode** to "`router`" if you use it.

X Window System daemons

xdm 'YES' or 'NO'. Runs the `xdm`(1) X display manager. These X daemons are available only with the optional X distribution of NetBSD.

xfs 'YES' or 'NO'. Runs the `xfs`(1) X11 font server, which supplies local X font files to X terminals.

NIS (YP) daemons

ypbind 'YES' or 'NO'. Runs `ypbind`(8), which lets NIS (YP) clients use information from a NIS server. Passes **ypbind_flags**.

yppasswdd 'YES' or 'NO'. Runs `yppasswdd`(8), which allows remote NIS users to update password on master server. Passes **yppasswdd_flags**.

ypserv 'YES' or 'NO'. Runs `ypserv`(8), the NIS (YP) server for distributing information from certain files in `/etc`. Passes **ypserv_flags**. The "`-d`" flag causes it to use DNS for lookups in `/etc/hosts` that fail.

NFS daemons and parameters

amd 'YES' or 'NO'. Runs `amd`(8), the automounter daemon, which automatically mounts NFS file systems whenever a file or directory within that file system is accessed. Passes **amd_flags**.

amd_dir A string. The `amd`(8) mount directory. Used only if **amd** is set to 'YES'.

lockd 'YES' or 'NO'. Runs `rpc.lockd`(8) if **nfs_server** and/or **nfs_client** are set to 'YES'. Passes **lockd_flags**.

mountd 'YES' or 'NO'. Runs `mountd`(8) and passes **mountd_flags**.

nfs_client 'YES' or 'NO'. The number of local NFS asynchronous I/O server is now controlled via `sysctl`(8).

nfs_server 'YES' or 'NO'. Sets up a host to be a NFS server by running `nfsd`(8) and passing **nfsd_flags**.

statd 'YES' or 'NO'. Runs `rpc.statd`(8), a status monitoring daemon used when `rpc.lockd`(8) is running, if **nfs_server** and/or **nfs_client** are set to 'YES'. Passes **statd_flags**.

Bluetooth configuration and daemons

btattach 'YES' or 'NO'. Attach serial bluetooth interfaces as listed in the configuration file `/etc/bluetooth/btdevctl.conf`.

btconfig 'YES' or 'NO'. Configure bluetooth devices. If the **btconfig_devices** variable below is not specified, all devices known to the system will be configured. For each device, configuration arguments are first looked for in the **btconfig_{dev}** variable, otherwise the value of the **btconfig_args** variable will be used, and if that is not specified the default string is 'enable'.

btconfig_devices An optional space separated list of bluetooth devices to be configured at boot time.

btconfig_args An optional string, containing default arguments for bluetooth devices to be configured.

btdevctl 'YES' or 'NO'. Configure Bluetooth devices as listed in the configuration file `/etc/bluetooth/btdevctl.conf`.

bthcid 'YES' or 'NO'. Runs `bthcid`(8), the Bluetooth HCI daemon, which manages link keys and PIN codes for Bluetooth links. Passes **bthcid_flags**.

sdpd 'YES' or 'NO'. Runs the Service Discovery Profile daemon, `sdpd`(8). Passes **sdpd_flags**.

Other daemons

isdnd 'YES' or 'NO'. Runs `isdnd`(8), the isdn4bsd ISDN connection management daemon. Passes **isdnd_flags**.

isdn_autoupdown 'YES' or 'NO'. Set all configured ISDN interfaces to "up". If **isdn_interfaces** is not blank, only the listed interfaces will be modified. Used only if **isdnd** is set to 'YES'.

kdc 'YES' or 'NO'. Runs the `kdc`(8) Kerberos v4 and v5 server. This should be run on Kerberos master and slave servers.

rwhod 'YES' or 'NO'. Runs `rwhod`(8) to support the `rwho`(1) and `ruptime`(1) commands.

Hardware daemons

apmd 'YES' or 'NO'. Runs `apmd`(8) and passes **apmd_flags**.

irdaattach 'YES' or 'NO'. Runs `irdaattach`(8) and passes **irdaattach_flags**.

moused 'YES' or 'NO'. Runs `moused`(8), to pass serial mouse data to the wscons mouse mux. Passes **moused_flags**.

poffd 'YES' or 'NO'. Runs `poffd`(8) x68k shutdown daemon (only for NetBSD/x68k). Passes **poffd_flags**.

screenblank 'YES' or 'NO'. Runs `screenblank`(1) and passes **screenblank_flags**.

wscons 'YES' or 'NO'. Configures the `wscons`(4) console driver, from the configuration file `/etc/wscons.conf`.

wsmoused 'YES' or 'NO'. Runs `wsmoused`(8), to provide copy and paste text support in wscons displays. Passes **wsmoused_flags**.

FILES

`/etc/rc.conf` The file **`rc.conf`** resides in `/etc`.

`/etc/defaults/rc.conf` Default settings for **`rc.conf`**, sourced by **`rc.conf`** before the end-user configuration section.

`/etc/rc.conf.d/`*`foo`* *`foo`*-specific **`rc.conf`** overrides.

SEE ALSO

`boot`(8), `rc`(8), `rc.d`(8), `rc.subr`(8), `rcorder`(8)

HISTORY

The **`rc.conf`** file appeared in NetBSD 1.3.

NAME

rcsfile − format of RCS file

DESCRIPTION

An RCS file's contents are described by the grammar below.

The text is free format: space, backspace, tab, newline, vertical tab, form feed, and carriage return (collectively, *white space*) have no significance except in strings. However, white space cannot appear within an id, num, or sym, and an RCS file must end with a newline.

Strings are enclosed by **@**. If a string contains a **@**, it must be doubled; otherwise, strings can contain arbitrary binary data.

The meta syntax uses the following conventions: '|' (bar) separates alternatives; '{' and '}' enclose optional phrases; '{' and '}*' enclose phrases that can be repeated zero or more times; '{' and '}+' enclose phrases that must appear at least once and can be repeated; Terminal symbols are in **boldface**; nonterminal symbols are in *italics*.

rcstext ::= *admin* {*delta*}* *desc* {*deltatext*}*

admin ::= **head** {*num*}**;**
{ **branch** {*num*}**;** }
access {*id*}***;**
symbols {*sym* **:** *num*}***;**
locks {*id* **:** *num*}***;** {**strict** **;**}
{ **comment** {*string*}**;** }
{ **expand** {*string*}**;** }
{ *newphrase* }*

delta ::= *num*
date *num***;**
author *id***;**
state {*id*}**;**
branches {*num*}***;**
next {*num*}**;**
{ *newphrase* }*

desc ::= **desc** *string*

deltatext ::= *num*
log *string*
{ *newphrase* }*
text *string*

num ::= {*digit* | **.**}+

digit ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

id ::= {*num*} *idchar* {*idchar* | *num*}*

sym ::= {*digit*}* *idchar* {*idchar* | *digit*}*

idchar ::= any visible graphic character except *special*

special ::= **$** | **,** | **.** | **:** | **;** | **@**

string ::= **@**{any character, with **@** doubled}***@**

newphrase ::= *id word** **;**

word ::= *id* | *num* | *string* | **:**

Identifiers are case sensitive. Keywords are in lower case only. The sets of keywords and identifiers can overlap. In most environments RCS uses the ISO 8859/1 encoding: visible graphic characters are codes 041−176 and 240−377, and white space characters are codes 010−015 and 040.

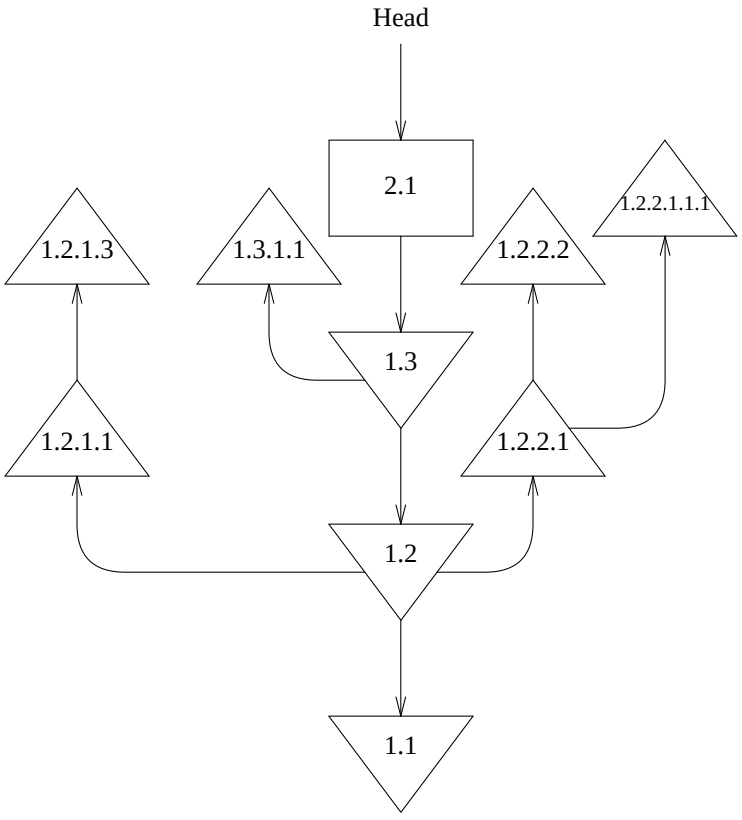
Dates, which appear after the **date** keyword, are of the form *Y.mm.dd.hh.mm.ss*, where *Y* is the year, *mm* the month (01−12), *dd* the day (01−31), *hh* the hour (00−23), *mm* the minute (00−59), and *ss* the second (00−60). *Y* contains just the last two digits of the year for years from 1900 through 1999, and all the digits of years thereafter. Dates use the Gregorian calendar; times use UTC.

The *newphrase* productions in the grammar are reserved for future extensions to the format of RCS files. No *newphrase* will begin with any keyword already in use.

The *delta* nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the trunk, and are linked through the **next** field in order of decreasing numbers. The **head** field in the *admin* node points to the head of that sequence (i.e., contains the highest pair). The **branch** node in the admin node indicates the default branch (or revision) for most RCS operations. If empty, the default branch is the highest branch on the trunk.

All *delta* nodes whose numbers consist of 2*n* fields (*n*2) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first 2*n*−1 number fields are identical are linked through the **next** field in order of increasing numbers. For each such sequence, the *delta* node whose number is identical to the first 2*n*−2 number fields of the deltas on that sequence is called the branchpoint. The **branches** field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

The following diagram shows an example of an RCS file's organization.



IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Manual Page Revision: ; Release Date: .

SEE ALSO

rcsintro(1), ci(1), co(1), ident(1), rcs(1), rcsclean(1), rcsdiff(1), rcsmerge(1), rlog(1)

Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

NAME

regexp_table − format of Postfix regular expression tables

SYNOPSIS

postmap -q "*string***" regexp:/etc/postfix/***filename*

postmap -q - regexp:/etc/postfix/*filename* <*inputfile*

DESCRIPTION

The Postfix mail system uses optional tables for address rewriting, mail routing, or access control. These tables are usually in **dbm** or **db** format.

Alternatively, lookup tables can be specified in POSIX regular expression form. In this case, each input is compared against a list of patterns. When a match is found, the corresponding result is returned and the search is terminated.

To find out what types of lookup tables your Postfix system supports use the "**postconf -m**" command.

To test lookup tables, use the "**postmap -q**" command as described in the SYNOPSIS above.

COMPATIBILITY

With Postfix version 2.2 and earlier specify "**postmap -fq**" to query a table that contains case sensitive patterns. Patterns are case insensitive by default.

TABLE FORMAT

The general form of a Postfix regular expression table is:

/*pattern*/*flags result*
: When *pattern* matches the input string, use the corresponding *result* value.

!/*pattern*/*flags result*
: When *pattern* does **not** match the input string, use the corresponding *result* value.

if /*pattern*/*flags*

endif
: Match the input string against the patterns between **if** and **endif**, if and only if that same input string also matches *pattern*. The **if**..**endif** can nest.

  Note: do not prepend whitespace to patterns inside **if**..**endif**.

  This feature is available in Postfix 2.1 and later.

if !/*pattern*/*flags*

endif
: Match the input string against the patterns between **if** and **endif**, if and only if that same input string does **not** match *pattern*. The **if**..**endif** can nest.

  Note: do not prepend whitespace to patterns inside **if**..**endif**.

  This feature is available in Postfix 2.1 and later.

blank lines and comments
: Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

multi-line text
: A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

Each pattern is a POSIX regular expression enclosed by a pair of delimiters. The regular expression syntax is documented in **re_format**(7) with 4.4BSD, in **regex**(5) with Solaris, and in **regex**(7) with Linux. Other systems may use other document names.

The expression delimiter can be any character, except whitespace or characters that have special meaning (traditionally the forward slash is used). The regular expression can contain whitespace.

By default, matching is case-insensitive, and newlines are not treated as special characters. The behavior is controlled by flags, which are toggled by appending one or more of the following characters after the pattern:

i (default: on)
Toggles the case sensitivity flag. By default, matching is case insensitive.

x (default: on)
Toggles the extended expression syntax flag. By default, support for extended expression syntax is enabled.

m (default: off)
Toggle the multi-line mode flag. When this flag is on, the ^ and **$** metacharacters match immediately after and immediately before a newline character, respectively, in addition to matching at the start and end of the input string.

TABLE SEARCH ORDER

Patterns are applied in the order as specified in the table, until a pattern is found that matches the input string.

Each pattern is applied to the entire input string. Depending on the application, that string is an entire client hostname, an entire client IP address, or an entire mail address. Thus, no parent domain or parent network search is done, and *user@domain* mail addresses are not broken up into their *user* and *domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

TEXT SUBSTITUTION

Substitution of substrings from the matched expression into the result string is possible using $1, $2, etc.; specify $$ to produce a $ character as output. The macros in the result string may need to be written as ${n} or $(n) if they aren't followed by whitespace.

Note: since negated patterns (those preceded by **!**) return a result when the expression does not match, substitutions are not available for negated patterns.

EXAMPLE SMTPD ACCESS MAP

```
# Disallow sender-specified routing. This is a must if you relay mail
# for other domains.
/[%!@].*[%!@]/       550 Sender-specified routing rejected

# Postmaster is OK, that way they can talk to us about how to fix
# their problem.
/^postmaster@/       OK

# Protect your outgoing majordomo exploders
if !/^owner-/
/^(.*)-outgoing@(.*)$/   550 Use ${1}@${2} instead
endif
```

EXAMPLE HEADER FILTER MAP

```
# These were once common in junk mail.
/^Subject: make money fast/     REJECT
/^To: friend@public\.com/       REJECT
```

EXAMPLE BODY FILTER MAP

```
# First skip over base 64 encoded text to save CPU cycles.
~^[[:alnum:]+/]{60,}$~          OK
```

```
# Put your own body patterns here.
```

SEE ALSO

postmap(1), Postfix lookup table manager
pcre_table(5), format of PCRE tables
cidr_table(5), format of CIDR tables

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview

AUTHOR(S)

The regexp table lookup code was originally written by:
LaMont Jones
lamont@hp.com

That code was based on the PCRE dictionary contributed by:
Andrew McNamara
andrewm@connect.com.au
connect.com.au Pty. Ltd.
Level 3, 213 Miller St
North Sydney, NSW, Australia

Adopted and adapted by:
Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

relocated − Postfix relocated table format

SYNOPSIS

postmap /etc/postfix/relocated

DESCRIPTION

The optional **relocated**(5) table provides the information that is used in "user has moved to *new_location*" bounce messages.

Normally, the **relocated**(5) table is specified as a text file that serves as input to the **postmap**(1) command. The result, an indexed file in **dbm** or **db** format, is used for fast searching by the mail system. Execute the command "**postmap /etc/postfix/relocated**" to rebuild an indexed file after changing the corresponding relocated table.

When the table is provided via other means such as NIS, LDAP or SQL, the same lookups are done as for ordinary indexed files.

Alternatively, the table can be provided as a regular-expression map where patterns are given as regular expressions, or lookups can be directed to TCP-based server. In those case, the lookups are done in a slightly different way as described below under "REGULAR EXPRESSION TABLES" or "TCP-BASED TABLES".

Table lookups are case insensitive.

CASE FOLDING

The search string is folded to lowercase before database lookup. As of Postfix 2.3, the search string is not case folded with database types such as regexp: or pcre: whose lookup fields can match both upper and lower case.

TABLE FORMAT

The input format for the **postmap**(1) command is as follows:

- An entry has one of the following form:

  pattern *new_location*

  Where *new_location* specifies contact information such as an email address, or perhaps a street address or telephone number.

- Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

- A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

TABLE SEARCH ORDER

With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, patterns are tried in the order as listed below:

user@domain
: Matches *user@domain*. This form has precedence over all other forms.

user
: Matches *user@site* when *site* is $**myorigin**, when *site* is listed in $**mydestination**, or when *site* is listed in $**inet_interfaces** or $**proxy_interfaces**.

@domain
: Matches other addresses in *domain*. This form has the lowest precedence.

ADDRESS EXTENSION

When a mail address localpart contains the optional recipient delimiter (e.g., *user+foo@domain*), the lookup order becomes: *user+foo@domain*, *user@domain*, *user+foo*, *user*, and *@domain*.

REGULAR EXPRESSION TABLES

This section describes how the table lookups change when the table is given in the form of regular expressions or when lookups are directed to a TCP-based server. For a description of regular expression lookup table syntax, see **regexp_table**(5) or **pcre_table**(5). For a description of the TCP client/server table lookup protocol, see **tcp_table**(5). This feature is not available up to and including Postfix version 2.4.

Each pattern is a regular expression that is applied to the entire address being looked up. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Patterns are applied in the order as specified in the table, until a pattern is found that matches the search string.

Results are the same as with indexed file lookups, with the additional feature that parenthesized substrings from the pattern can be interpolated as **$1**, **$2** and so on.

TCP-BASED TABLES

This section describes how the table lookups change when lookups are directed to a TCP-based server. For a description of the TCP client/server lookup protocol, see **tcp_table**(5). This feature is not available up to and including Postfix version 2.4.

Each lookup operation uses the entire address once. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Results are the same as with indexed file lookups.

BUGS

The table format does not understand quoting conventions.

CONFIGURATION PARAMETERS

The following **main.cf** parameters are especially relevant. The text below provides only a parameter summary. See **postconf**(5) for more details including examples.

relocated_maps
List of lookup tables for relocated users or sites.

Other parameters of interest:

inet_interfaces
The network interface addresses that this system receives mail on. You need to stop and start Postfix when this parameter changes.

mydestination
List of domains that this mail system considers local.

myorigin
The domain that is appended to locally-posted mail.

proxy_interfaces
Other interfaces that this machine receives mail on by way of a proxy agent or network address translator.

SEE ALSO

trivial-rewrite(8), address resolver
postmap(1), Postfix lookup table manager
postconf(5), configuration parameters

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview
ADDRESS_REWRITING_README, address rewriting guide

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

remote — remote host description file

DESCRIPTION

The systems known by tip(1) and their attributes are stored in an ASCII file which is structured somewhat like the termcap(5) file. Each line in the file provides a description for a single *system*. Fields are separated by a colon (":"). Lines ending in a \ character with an immediately following newline are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an '=' sign indicates a string value follows. A field name followed by a '#' sign indicates a following numeric value.

Entries named "tip∗" and "cu∗" are used as default entries by tip(1), and the cu(1) interface to tip(1), as follows. When tip(1) is invoked with only a phone number, it looks for an entry of the form "tip300", where 300 is the baud rate with which the connection is to be made. When the cu(1) interface is used, entries of the form "cu300" are used.

CAPABILITIES

Capabilities are either strings (str), numbers (num), or boolean flags (bool). A string capability is specified by *capability=value*; for example, "dv=/dev/harris". A numeric capability is specified by *capability#value*; for example, "xa#99". A boolean capability is specified by simply listing the capability.

at (str) Auto call unit type.

br (num) The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is 300 baud.

cm (str) An initial connection message to be sent to the remote host. For example, if a host is reached through a port selector, this might be set to the appropriate sequence required to switch to the host.

cu (str) Call unit if making a phone call. Default is the same as the 'dv' field.

dc (bool) This host is directly connected, and tip should not expect carrier detect to be high, nor should it exit if carrier detect drops.

di (str) Disconnect message sent to the host when a disconnect is requested by the user.

du (bool) This host is on a dial-up line.

dv (str) UNIX device(s) to open to establish a connection. If this file refers to a terminal line, tip(1) attempts to perform an exclusive open on the device to ensure only one user at a time has access to the port.

el (str) Characters marking an end-of-line. The default is NULL. '~' escapes are only recognized by tip(1) after one of the characters in 'el', or after a carriage-return.

fs (str) Frame size for transfers. The default frame size is equal to BUFSIZ.

hd (bool) The host uses half-duplex communication, local echo should be performed.

hf (bool) Use hardware (RTS/CTS) flow control.

ie (str) Input end-of-file marks. The default is NULL.

oe (str) Output end-of-file string. The default is NULL. When tip(1) is transferring a file, this string is sent at end-of-file.

pa (str) The type of parity to use when sending data to the host. This may be one of **even**, **odd**, **none**, **zero** (always set bit 8 to zero), **one** (always set bit 8 to one). The default is even parity.

pn (str) Telephone number(s) for this host. If the telephone number field contains an @ sign, `tip`(1) searches the file `/etc/phones` file for a list of telephone numbers; see `phones`(5).

tc (str) Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature:

```
UNIX-1200:\
:dv=/dev/cau0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:
arpavax|ax:\
:pn=7654321%:tc=UNIX-1200
```

FILES

`/etc/remote` The **remote** host description file resides in `/etc`.

SEE ALSO

`tip`(1), `phones`(5)

HISTORY

The **remote** file format appeared in 4.2BSD.

NAME

resolv.conf — resolver configuration file

DESCRIPTION

The **resolv.conf** file specifies how the resolver(3) routines in the C library (which provide access to the Internet Domain Name System) should operate. The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of keywords with values that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine, the domain name is determined from the host name, and the domain search path is constructed from the domain name.

The different configuration options are:

nameserver IPv4 address (in dot notation) or IPv6 address (in hex-and-colon notation) of a name server that the resolver should query. Scoped IPv6 address notation is accepted as well (see inet6(4) for details). Up to MAXNS (currently 3) name servers may be listed, one per keyword. If there are multiple servers, the resolver library queries them in the order listed. If no **nameserver** entries are present, the default is to use the name server on the local machine. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made).

domain Local domain name. Most queries for names within this domain can use short names relative to the local domain. If no **domain** entry is present, the domain is determined from the local host name returned by gethostname(3); the domain part is taken to be everything after the first '.'. Finally, if the host name does not contain a domain part, the root domain is assumed.

lookup This keyword is now ignored: its function has been superseded by features of nsswitch.conf(5).

search Search list for host-name lookup. The search list is normally determined from the local domain name; by default, it begins with the local domain name, then successive parent domains that have at least two components in their names. This may be changed by listing the desired domain search path following the **search** keyword with spaces or tabs separating the names. Most resolver queries will be attempted using each component of the search path in turn until a match is found. Note that this process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local, and that queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains with a total of 1024 characters.

sortlist Sortlist allows addresses returned by gethostbyname to be sorted. A sortlist is specified by IP address netmask pairs. The netmask is optional and defaults to the natural netmask of the net. The IP address and optional network pairs are separated by slashes. Up to 10 pairs may be specified, ie.

sortlist 130.155.160.0/255.255.240.0 130.155.0.0

options Options allows certain internal resolver variables to be modified. The syntax is:

options option ...

where option is one of the following:

debug enable debugging information, by setting RES_DEBUG in _res.options (see `resolver`(3)).

edns0 attach OPT pseudo-RR for ENDS0 extension specified in RFC 2671, to inform DNS server of our receive buffer size. The option will allow DNS servers to take advantage of non-default receive buffer size, and to send larger replies. DNS query packets with EDNS0 extension is not compatible with non-EDNS0 DNS servers. The option must be used only when all the DNS servers listed in **nameserver** lines are able to handle EDNS0 extension.

inet6 enable support for IPv6-only applications, by setting RES_USE_INET6 in _res.options (see `resolver`(3)). The option is meaningful with certain kernel configuration only and use of this option is discouraged.

insecure1 Do not require IP source address on the reply packet to be equal to the servers' address.

insecure2 Do not check if the query section of the reply packet is equal to that of the query packet. For testing purposes only.

ndots:n sets a threshold for the number of dots which must appear in a name given to res_query (see `resolver`(3)) before an initial absolute query will be made. The default for n is 1, meaning that if there are any dots in a name, the name will be tried first as an absolute name before any search list elements are appended to it.

The **domain** and **search** keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance will override.

The **search** keyword of a system's `resolv.conf` file can be overridden on a per-process basis by setting the environment variable `LOCALDOMAIN` to a space-separated list of search domains.

The **options** keyword of a system's `resolv.conf` file can be amended on a per-process basis by setting the environment variable `RES_OPTIONS` to a space-separated list of resolver options as explained above.

The keyword and value must appear on a single line, and the keyword (e.g. **nameserver**) must start the line. The value follows the keyword, separated by white space.

FILES

`/etc/resolv.conf` The file **`resolv.conf`** resides in `/etc`.

SEE ALSO

`gethostbyname`(3), `resolver`(3), `nsswitch.conf`(5), `hostname`(7), `named`(8)

Name Server Operations Guide for BIND.

HISTORY

The **`resolv.conf`** file format appeared in 4.3BSD.

NAME

rndc.conf − rndc configuration file

SYNOPSIS

rndc.conf

DESCRIPTION

rndc.conf is the configuration file for **rndc**, the BIND 9 name server control utility. This file has a similar structure and syntax to *named.conf*. Statements are enclosed in braces and terminated with a semi−colon. Clauses in the statements are also semi−colon terminated. The usual comment styles are supported:

C style: /* */

C++ style: // to end of line

Unix style: # to end of line

rndc.conf is much simpler than *named.conf*. The file uses three statements: an options statement, a server statement and a key statement.

The **options** statement contains five clauses. The **default−server** clause is followed by the name or address of a name server. This host will be used when no name server is given as an argument to **rndc**. The **default−key** clause is followed by the name of a key which is identified by a **key** statement. If no **keyid** is provided on the rndc command line, and no **key** clause is found in a matching **server** statement, this default key will be used to authenticate the server's commands and responses. The **default−port** clause is followed by the port to connect to on the remote name server. If no **port** option is provided on the rndc command line, and no **port** clause is found in a matching **server** statement, this default port will be used to connect. The **default−source−address** and **default−source−address−v6** clauses which can be used to set the IPv4 and IPv6 source addresses respectively.

After the **server** keyword, the server statement includes a string which is the hostname or address for a name server. The statement has three possible clauses: **key**, **port** and **addresses**. The key name must match the name of a key statement in the file. The port number specifies the port to connect to. If an **addresses** clause is supplied these addresses will be used instead of the server name. Each address can take a optional port. If an **source−address** or **source−address−v6** of supplied then these will be used to specify the IPv4 and IPv6 source addresses respectively.

The **key** statement begins with an identifying string, the name of the key. The statement has two clauses. **algorithm** identifies the encryption algorithm for **rndc** to use; currently only HMAC−MD5 is supported. This is followed by a secret clause which contains the base−64 encoding of the algorithm's encryption key. The base−64 string is enclosed in double quotes.

There are two common ways to generate the base−64 string for the secret. The BIND 9 program **rndc−confgen** can be used to generate a random key, or the **mmencode** program, also known as **mimencode**, can be used to generate a base−64 string from known input. **mmencode** does not ship with BIND 9 but is available on many systems. See the EXAMPLE section for sample command lines for each.

EXAMPLE

```
options {
  default−server  localhost;
  default−key     samplekey;
};

server localhost {
  key             samplekey;
};

server testserver {
  key             testkey;
  addresses       { localhost port 5353; };
```

```
};

key samplekey {
  algorithm     hmac−md5;
  secret        "6FMfj43Osz4lyb24OIe2iGEz9lf1llJO+lz";
};

key testkey {
  algorithm    hmac−md5;
  secret                "R3HI8P6BKw9ZwXwN3VZKuQ==";
}
```

In the above example, **rndc** will by default use the server at localhost (127.0.0.1) and the key called samplekey. Commands to the localhost server will use the samplekey key, which must also be defined in the server's configuration file with the same name and secret. The key statement indicates that samplekey uses the HMAC−MD5 algorithm and its secret clause contains the base−64 encoding of the HMAC−MD5 secret enclosed in double quotes.

If **rndc −s testserver** is used then **rndc** will connect to server on localhost port 5353 using the key testkey.

To generate a random secret with **rndc−confgen**:

rndc−confgen

A complete *rndc.conf* file, including the randomly generated key, will be written to the standard output. Commented out **key** and **controls** statements for *named.conf* are also printed.

To generate a base−64 secret with **mmencode**:

echo "known plaintext for a secret" | mmencode

NAME SERVER CONFIGURATION

The name server must be configured to accept rndc connections and to recognize the key specified in the *rndc.conf* file, using the controls statement in *named.conf*. See the sections on the **controls** statement in the BIND 9 Administrator Reference Manual for details.

SEE ALSO

rndc(8), **rndc−confgen**(8), **mmencode**(1), BIND 9 Administrator Reference Manual.

AUTHOR

Internet Systems Consortium

COPYRIGHT

NAME

route.conf — static routes config file

DESCRIPTION

The **route.conf** file is read by the staticroute rc.d script during system start-up and shutdown, and is intended for adding and removing static routes.

FILE FORMAT

Lines starting with a hash (‘#’) are comments and ignored. Lines starting with a plus sign (‘+’) are run during start-up, while lines starting with a minus sign (‘−’) are run during system shutdown. All other lines are passed to route(8). During start-up, they are passed behind a “**route add −**” command and during shutdown behind a “**route delete −**” command.

FILES

/etc/route.conf The **route.conf** file resides in /etc.

/etc/rc.d/staticroute
rc.d(8) script that parses **route.conf**.

EXAMPLES

In this example, if the staticroute script is enabled in rc.conf(5), IP forwarding is turned on during start-up, and a static route added for 192.168.2.0. During system shutdown, the route is removed and IP forwarding turned off.

```
# Turn on/off IP forwarding.
+sysctl -w net.inet.ip.forwarding=1
-sysctl -w net.inet.ip.forwarding=0
net 192.168.2.0 -netmask 255.255.255.0 192.168.150.2
```

SEE ALSO

rc.conf(5), rc(8), route(8)

NAME

rpc — rpc program number data base

SYNOPSIS

/etc/rpc

DESCRIPTION

The **rpc** file is a local source containing user readable names that can be used in place of RPC program numbers.

The rpc file has one line for each RPC program name. The line has the following format:

Items are separated by any number of blanks and/or tab characters. A hash ("#") indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

FILES

/etc/nsswitch.conf

EXAMPLES

Below is an example of an RPC database:

```
#
#       rpc
#
rpcbind         100000  portmap sunrpc portmapper
rusersd         100002  rusers
nfs             100003  nfsprog
mountd          100005  mount   showmount
walld           100008  rwall   shutdown
sprayd          100012  spray
llockmgr        100020
nlockmgr        100021
status          100024
bootparam       100026
keyserv         100029  keyserver
```

SEE ALSO

getrpcent(3)

NAME

rtadvd.conf — config file for router advertisement daemon

DESCRIPTION

This file describes how the router advertisement packets must be constructed for each of the interfaces.

As described in rtadvd(8), you do not have to set this configuration file up at all, unless you need some special configurations. You may even omit the file as a whole. In such cases, the **rtadvd** daemon will automatically configure itself using default values specified in the specification.

It obeys the famous termcap(5) file format. Each line in the file describes a network interface. Fields are separated by a colon (':'), and each field contains one capability description. Lines may be concatenated by the '\' character. The comment marker is the '#' character.

CAPABILITIES

Capabilities describe the value to be filled into ICMPv6 router advertisement messages and to control rtadvd(8) behavior. Therefore, you are encouraged to read IETF neighbor discovery documents if you would like to modify the sample configuration file.

Note that almost all items have default values. If you omit an item, the default value of the item will be used.

There are two items which control the interval of sending router advertisements. These items can be omitted, then **rtadvd** will use the default values.

maxinterval
(num) The maximum time allowed between sending unsolicited multicast router advertisements (unit: seconds). The default value is 600. Its value must be no less than 4 seconds and no greater than 1800 seconds.

mininterval
(num) The minimum time allowed between sending unsolicited multicast router advertisements (unit: seconds). The default value is the one third of value of **maxinterval**. Its value must be no less than 3 seconds and no greater than .75 * the value of **maxinterval**.

The following items are for ICMPv6 router advertisement message header. These items can be omitted, then **rtadvd** will use the default values.

chlim (num) The value for Cur Hop Limit field. The default value is 64.

raflags
(str or num) A 8-bit flags field in router advertisement message header. This field can be specified either as a case-sensitive string or as an integer. A sting consists of characters each of which corresponds to a particular flag bit(s). An integer should be the logical OR of all enabled bits. Bit 7 ('m' or 0x80) means Managed address configuration flag bit, and Bit 6 ('o' or 0x40) means Other stateful configuration flag bit. Bit 4 (0x10) and Bit 3 (0x08) are used to encode router preference. Bits 01 (or 'h') means high, 00 means medium, and 11 (or 'l') means low. Bits 10 is reserved, and must not be specified. There is no character to specify the medium preference explicitly. The default value of the entire flag is 0 (or a null string,) which means no additional configuration methods, and the medium router preference.

rltime (num) Router lifetime field (unit: seconds). The value must be either zero or between the value of **maxinterval** and 9000. When **rtadvd** runs on a host, this value must explicitly set 0 on all the advertising interfaces as described in rtadvd(8). The default value is 1800.

rtime (num) Reachable time field (unit: milliseconds). The default value is 0, which means unspecified by this router.

retrans
(num) Retrans Timer field (unit: milliseconds). The default value is 0, which means unspecified by this router.

The following items are for ICMPv6 prefix information option, which will be attached to router advertisement header. These items can be omitted, then **rtadvd** will automatically get appropriate prefixes from the kernel's routing table, and advertise the prefixes with the default parameters. Keywords other than **clockskew** can be augmented with a number, like "prefix2", to specify multiple prefixes.

clockskew
(num) Time skew to adjust link propagation delays and clock skews between routers on the link (unit: seconds). This value is used in consistency check for locally-configured and advertised prefix lifetimes, and has its meaning when the local router configures a prefix on the link with a lifetime that decrements in real time. If the value is 0, it means the consistency check will be skipped for such prefixes. The default value is 0.

prefixlen
(num) Prefix length field. The default value is 64.

pinfoflags
(str or num) A 8-bit flags field in prefix information option. This field can be specified either as a case-sensitive string or as an integer. A sting consists of characters each of which corresponds to a particular flag bit(s). An integer should be the logical OR of all enabled bits. Bit 7 (`'l' or 0x80`) means On-link flag bit, and Bit 6 (`'a' or 0x40`) means Autonomous address-configuration flag bit. The default value is "la" or 0xc0, i.e., both bits are set.

addr (str) The address filled into Prefix field. Since ":" is used for termcap(5) file format as well as IPv6 numeric address, the field MUST be quoted by doublequote character.

vltime (num) Valid lifetime field (unit: seconds). The default value is 2592000 (30 days).

vltimedecr
(bool) This item means the advertised valid lifetime will decrement in real time, which is disabled by default.

pltime (num) Preferred lifetime field (unit: seconds). The default value is 604800 (7 days).

pltimedecr
(bool) This item means the advertised preferred lifetime will decrement in real time, which is disabled by default.

The following item is for ICMPv6 MTU option, which will be attached to router advertisement header. This item can be omitted, then **rtadvd** will use the default value.

mtu (num or str) MTU (maximum transmission unit) field. If 0 is specified, it means that the option will not be included. The default value is 0. If the special string "auto" is specified for this item, MTU option will be included and its value will be set to the interface MTU automatically.

The following item controls ICMPv6 source link-layer address option, which will be attached to router advertisement header. As noted above, you can just omit the item, then **rtadvd** will use the default value.

nolladdr
(bool) By default (if **nolladdr** is not specified), rtadvd(8) will try to get link-layer address for the interface from the kernel, and attach that in source link-layer address option. If this capability exists, rtadvd(8) will not attach source link-layer address option to router advertisement packets.

The following items are for ICMPv6 route information option, which will be attached to router advertisement header. These items are optional. Each items can be augmented with number, like "rtplen2", to specify multiple routes.

rtprefix
(str) The prefix filled into the Prefix field of route information option. Since ":" is used for termcap(5) file format as well as IPv6 numeric address, the field MUST be quoted by double-quote character.

rtplen (num) Prefix length field in route information option. The default value is 64.

rtflags
(str or num) A 8-bit flags field in route information option. Currently only the preference values are defined. The notation is same as that of the raflags field. Bit 4 (0x10) and and Bit 3 (0x08) are used to encode the route preference for the route. The default value is 0x00, i.e. medium preference.

rtltime
(num) route lifetime field in route information option. (unit: seconds). Since the specification does not define the default value of this item, the value for this item should be specified by hand. However, **rtadvd** allows this item to be unspecified, and uses the router lifetime as the default value in such a case, just for compatibility with an old version of the program.

In the above list, each keyword beginning with "rt" could be replaced with the one beginning with "rtr" for backward compatibility reason. For example, **rtrplen** is accepted instead of **rtplen**. However, keywords that start with "rtr" have basically been obsoleted, and should not be used any more.

You can also refer one line from another by using **tc** capability. See termcap(5) for details on the capability.

EXAMPLES

As presented above, all of the advertised parameters have default values defined in specifications, and hence you usually do not have to set them by hand, unless you need special non-default values. It can cause interoperability problem if you use an ill-configured parameter.

To override a configuration parameter, you can specify the parameter alone. With the following configuration, rtadvd(8) overrides the router lifetime parameter for the ne0 interface.

```
ne0:\
        :rltime#0:
```

The following example manually configures prefixes advertised from the ef0 interface. The configuration must be used with the **-s** option to rtadvd(8).

```
ef0:\
        :addr="3ffe:501:ffff:1000::":prefixlen#64:
```

The following example presents the default values in an explicit manner. The configuration is provided just for reference purposes; YOU DO NOT NEED TO HAVE IT AT ALL.

```
default:\
        :chlim#64:raflags#0:rltime#1800:rtime#0:retrans#0:\
        :pinfoflags="la":vltime#2592000:pltime#604800:mtu#0:
ef0:\
        :addr="3ffe:501:ffff:1000::":prefixlen#64:tc=default:
```

SEE ALSO

`termcap`(5), `rtadvd`(8), `rtsol`(8)

Thomas Narten, Erik Nordmark and W. A. Simpson, “Neighbor Discovery for IP version 6 (IPv6)”, RFC 2461

Richard Draves, “Default Router Preferences and More-Specific Routes”, RFC 4191

HISTORY

The `rtadvd`(8) and the configuration file **`rtadvd.conf`** first appeared in WIDE Hydrangea IPv6 protocol stack kit.

NAME

security.conf — daily security check configuration file

DESCRIPTION

The **security.conf** file specifies which of the standard /etc/security services are performed. The /etc/security script is run, by default, every night from /etc/daily, on a NetBSD system, if configured do to so from /etc/daily.conf.

The variables described below can be set to "NO" to disable the test:

check_passwd This checks the /etc/master.passwd file for inconsistencies.

check_group This checks the /etc/group file for inconsistencies.

check_rootdotfiles This checks the root users startup files for sane settings of $PATH and umask. This test is not fail safe and any warning generated from this should be checked for correctness.

check_ftpusers This checks that the correct users are in the /etc/ftpusers file.

check_aliases This checks for security problems in the /etc/mail/aliases file. For backward compatibility, /etc/aliases will be checked as well if exists.

check_rhosts This checks for system and user rhosts files with "+" in them.

check_homes This checks that home directories are owned by the correct user, and have appropriate permissions.

check_varmail This checks that the correct user owns mail in /var/mail, and that the mail box has the right permissions.

check_nfs This checks that the /etc/exports file does not export filesystems to the world.

check_devices This checks for changes to devices and setuid files.

check_mtree This runs mtree(8) to ensure that the system is installed correctly. The following configuration files are checked:

/etc/mtree/special
: Default files to check.

/etc/mtree/special.local
: Local site additions and overrides.

/etc/mtree/DIR.secure
: Specification for the directory DIR.

check_disklabels Backup text copies of the disklabels of available disk drives into /var/backups/work/disklabel.XXX, and display any differences in those and the previous copies as per **check_changelist** below. If fdisk(8) is available on the current platform, the output of /sbin/fdisk for each available disk drive is stored in /var/backups/work/fdisk.XXX, and any differences displayed as per the disklabels.

check_pkgs This stores a list of all installed pkgs into /var/backups/work/pkgs and checks it for any changes.

check_changelist This determines a list of files from the contents of /etc/changelist, and the output of **mtree -D** for /etc/mtree/special and /etc/mtree/special.local. For each file in the list it compares the files with their backups in /var/backups/file.current and /var/backups/file.backup, and displays any differences found. The following mtree(8) **tags** modify how files are deter-

mined from `/etc/mtree/special` and `/etc/mtree/special.local`:

exclude The entry is ignored; no backups are made and the differences are not displayed. This includes dynamic or binary files such as `/var/run/utmp`.

nodiff The entry is backed up but the differences are not displayed because the contents of the file are sensitive. This includes files such as `/etc/master.passwd`.

The variables described below can be set to modify the tests:

check_homes_permit_usergroups
During the **check_homes** phase, allow the checked files to be group-writable if the group name is the same as the username.

check_devices_ignore_fstypes
Lists filesystem types to ignore during the **check_devices** phase. Prefixing the type with a '!' inverts the match. For example, `procfs !local` will ignore `procfs` type filesystems and filesystems that are not `local`.

check_devices_ignore_paths
Lists pathnames to ignore during the **check_devices** phase. Prefixing the path with a '!' inverts the match. For example, `/tftp` will ignore paths under `/tftp` while `!/home` will ignore paths that are not under `/home`.

check_mtree_follow_symlinks
During the **check_mtree** phase, instruct mtree to follow symbolic links. Please note, this may cause the **check_mtree** phase to report errors for entries for these symbolic links (i.e. of type=link in the mtree specification) as they will always appear to be plain files for the purposes of the check. `/etc/mtree/special.local` may be used to override the checks for the affected links.

check_passwd_nowarn_shells
If **check_passwd** is enabled, most warnings will be suppressed for entries whose shells are listed in this space-separated list. This is of particular value when those shells are not in `/etc/shells`.

check_passwd_nowarn_users
If **check_passwd** is enabled, suppress warnings for these users.

check_passwd_permit_nonalpha
If **check_passwd** is enabled, do not warn about login names which use non-alphanumeric characters.

check_passwd_permit_star
If **check_passwd** is enabled, do not warn about password fields set to "*". Note that the use of password fields such as "*ssh" is encouraged, instead.

max_grouplen If **check_group** is enabled, this determines the maximum permitted length of group names.

max_loginlen If **check_passwd** is enabled, this determines the maximum permitted length of login names.

backup_dir Change the backup directory from `/var/backup`.

diff_options Specify the options passed to `diff`(1) when it is invoked to show changes made to system files. Defaults to "-u", for unified-format context-diffs.

pkgdb_dir Change the pkg database directory from `/var/db/pkg` when **check_pkgs** is enabled.

backup_uses_rcs Use `rcs`(1) for maintaining backup copies of files noted in **check_devices**, **check_disklabels**, **check_pkgs**, and **check_changelist** instead of just keeping a current copy and a backup copy.

FILES

`/etc/defaults/security.conf`	defaults for /etc/security.conf
`/etc/security`	daily security check script
`/etc/security.conf`	daily security check configuration
`/etc/security.local`	local site additions to `/etc/security`

SEE ALSO

`daily.conf`(5)

HISTORY

The **`security.conf`** file appeared in NetBSD 1.3. The **check_disklabels** functionality was added in NetBSD 1.4. The **backup_uses_rcs** and **check_pkgs** features were added in NetBSD 1.6. **diff_options** appeared in NetBSD 2.0; prior to that, traditional-format (context free) diffs were generated.

NAME

services — service name data base

DESCRIPTION

The **services** file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single *item*; a slash ("/") is used to separate the port and protocol (e.g. "512/tcp"). A hash ("#") indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

The database in /var/db/services.db needs to be updated with `services_mkdb`(8) after changes to the services file have been applied.

FILES

`/etc/services` The **services** file resides in `/etc`.

SEE ALSO

`getservent`(3) `services_mkdb`(8)

HISTORY

The **services** file format appeared in 4.2BSD.

BUGS

A name server should be used instead of a static file.

NAME

shells — shell database

DESCRIPTION

The **shells** file contains a list of the shells on the system. It can be used in conjunction with the Hesiod domain 'shells', and the NIS map 'shells', as controlled by `nsswitch.conf`(5).

For each shell a single line should be present, consisting of the shell's path, relative to root.

A hash ("#") indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file. Blank lines are also ignored.

FILES

`/etc/shells` The **shells** file resides in `/etc`.

SEE ALSO

`chsh`(1), `getusershell`(3), `nsswitch.conf`(5)

HISTORY

The **shells** file format appeared in 4.3BSD−Tahoe.

The Hesiod and NIS support first appeared in NetBSD 1.4.

NAME

slapd-bdb, slapd-hdb − Berkeley DB backends to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **bdb** backend to **slapd**(8) is the recommended primary backend for a normal **slapd** database. It uses the Oracle Berkeley DB (BDB) package to store data. It makes extensive use of indexing and caching to speed data access.

hdb is a variant of the **bdb** backend that uses a hierarchical database layout which supports subtree renames. It is otherwise identical to the **bdb** behavior, and all the same configuration options apply.

It is noted that these options are intended to complement Berkeley DB configuration options set in the environment's **DB_CONFIG** file. See Berkeley DB documentation for details on **DB_CONFIG** configuration options. Where there is overlap, settings in **DB_CONFIG** take precedence.

CONFIGURATION

These **slapd.conf** options apply to the **bdb** and **hdb** backend database. That is, they must follow a "database bdb" or "database hdb" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

cachesize <*integer*>
Specify the size in entries of the in-memory entry cache maintained by the **bdb** or **hdb** backend database instance. The default is 1000 entries.

cachefree <*integer*>
Specify the number of entries to free from the entry cache when the cache reaches the **cachesize** limit. The default is 1 entry.

checkpoint <*kbyte*> <*min*>
Specify the frequency for checkpointing the database transaction log. A checkpoint operation flushes the database buffers to disk and writes a checkpoint record in the log. The checkpoint will occur if either <*kbyte*> data has been written or <*min*> minutes have passed since the last checkpoint. Both arguments default to zero, in which case they are ignored. When the <*min*> argument is non-zero, an internal task will run every <*min*> minutes to perform the checkpoint. See the Berkeley DB reference guide for more details.

cryptfile <*file*>
Specify the pathname of a file containing an encryption key to use for encrypting the database. Encryption is performed using Berkeley DB's implementation of AES. Note that encryption can only be configured before any database files are created, and changing the key can only be done after destroying the current database and recreating it. Encryption is not enabled by default, and some distributions of Berkeley DB do not support encryption.

cryptkey <*key*>
Specify an encryption key to use for encrypting the database. This option may be used when a separate *cryptfile* is not desired. Only one of **cryptkey** or **cryptfile** may be configured.

dbconfig <*Berkeley−DB−setting*>
Specify a configuration directive to be placed in the **DB_CONFIG** file of the database directory. The **dbconfig** directive is just a convenience to allow all necessary configuration to be set in the **slapd.conf** file. The options set using this directive will only be written to the **DB_CONFIG** file if no such file existed at server startup time, otherwise they are completely ignored. This allows one to set initial values without overwriting/destroying a **DB_CONFIG** file that was already customized through other means. This directive may be specified multiple times, as needed. For example:

```
dbconfig set_cachesize 0 1048576 0
dbconfig set_lg_bsize 2097152
```

dbnosync
Specify that on-disk database contents should not be immediately synchronized with in memory changes. Enabling this option may improve performance at the expense of data security. See the Berkeley DB reference guide for more details.

directory <*directory*>
Specify the directory where the BDB files containing this database and associated indexes live. A separate directory must be specified for each database. The default is **LOCALSTATEDIR/openldap-data**.

dirtyread
Allow reads of modified but not yet committed data. Usually transactions are isolated to prevent other operations from accessing uncommitted data. This option may improve performance, but may also return inconsistent results if the data comes from a transaction that is later aborted. In this case, the modified data is discarded and a subsequent search will return a different result.

dncachesize <*integer*>
Specify the maximum number of DNs in the in-memory DN cache. The default is twice the **cachesize**. Ideally this cache should be large enough to contain the DNs of every entry in the database.

idlcachesize <*integer*>
Specify the size of the in-memory index cache, in index slots. The default is zero. A larger value will speed up frequent searches of indexed entries. An **hdb** database needs a large **idlcachesize** for good search performance, typically three times the **cachesize** (entry cache size) or larger.

index {<*attrlist*>|**default**} [**pres**,**eq**,**approx**,**sub**,<*special*>]
Specify the indexes to maintain for the given attribute (or list of attributes). Some attributes only support a subset of indexes. If only an <*attr*> is given, the indices specified for **default** are maintained. Note that setting a default does not imply that all attributes will be indexed. Also, for best performance, an **eq** index should always be configured for the **objectClass** attribute.

A number of special index parameters may be specified. The index type **sub** can be decomposed into **subinitial**, **subany**, and **subfinal** indices. The special type **nolang** may be specified to disallow use of this index by language subtypes. The special type **nosubtypes** may be specified to disallow use of this index by named subtypes. Note: changing **index** settings in **slapd.conf**(5) requires rebuilding indices, see **slapindex**(8); changing **index** settings dynamically by LDAPModifying "cn=config" automatically causes rebuilding of the indices online in a background task.

linearindex
Tell **slapindex** to index one attribute at a time. By default, all indexed attributes in an entry are processed at the same time. With this option, each indexed attribute is processed individually, using multiple passes through the entire database. This option improves **slapindex** performance when the database size exceeds the **dbcache** size. When the **dbcache** is large enough, this option is not needed and will decrease performance. Also by default, **slapadd** performs full indexing and so a separate **slapindex** run is not needed. With this option, **slapadd** does no indexing and **slapindex** must be used.

lockdetect {**oldest**|**youngest**|**fewest**|**random**|**default**}
Specify which transaction to abort when a deadlock is detected. The default is **random**.

mode <*integer*>
Specify the file protection mode that newly created database index files should have. The default is 0600.

searchstack <*depth*>
Specify the depth of the stack used for search filter evaluation. Search filters are evaluated on a stack to accommodate nested AND / OR clauses. An individual stack is assigned to each server thread. The depth of the stack determines how complex a filter can be evaluated without requiring any additional memory allocation. Filters that are nested deeper than the search stack depth will cause a separate stack to be allocated for that particular search operation. These allocations can

have a major negative impact on server performance, but specifying too much stack will also consume a great deal of memory. Each search stack uses 512K bytes per level. The default stack depth is 16, thus 8MB per thread is used.

shm_key <*integer*>
Specify a key for a shared memory BDB environment. By default the BDB environment uses memory mapped files. If a non-zero value is specified, it will be used as the key to identify a shared memory region that will house the environment.

ACCESS CONTROL

The **bdb** and **hdb** backends honor access control semantics as indicated in **slapd.access**(5).

FILES

ETCDIR/slapd.conf
default **slapd** configuration file

DB_CONFIG
Berkeley DB configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **slapadd**(8), **slapcat**(8), **slapindex**(8), Berkeley DB documentation.

ACKNOWLEDGEMENTS

Originally begun by Kurt Zeilenga. Caching mechanisms originally designed by Jong-Hyuk Choi. Completion and subsequent work, as well as back-hdb, by Howard Chu.

NAME

slapd-bdb, slapd-hdb − Berkeley DB backends to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **bdb** backend to **slapd**(8) is the recommended primary backend for a normal **slapd** database. It uses the Oracle Berkeley DB (BDB) package to store data. It makes extensive use of indexing and caching to speed data access.

hdb is a variant of the **bdb** backend that uses a hierarchical database layout which supports subtree renames. It is otherwise identical to the **bdb** behavior, and all the same configuration options apply.

It is noted that these options are intended to complement Berkeley DB configuration options set in the environment's **DB_CONFIG** file. See Berkeley DB documentation for details on **DB_CONFIG** configuration options. Where there is overlap, settings in **DB_CONFIG** take precedence.

CONFIGURATION

These **slapd.conf** options apply to the **bdb** and **hdb** backend database. That is, they must follow a "database bdb" or "database hdb" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

cachesize <*integer*>
Specify the size in entries of the in-memory entry cache maintained by the **bdb** or **hdb** backend database instance. The default is 1000 entries.

cachefree <*integer*>
Specify the number of entries to free from the entry cache when the cache reaches the **cachesize** limit. The default is 1 entry.

checkpoint <*kbyte*> <*min*>
Specify the frequency for checkpointing the database transaction log. A checkpoint operation flushes the database buffers to disk and writes a checkpoint record in the log. The checkpoint will occur if either <*kbyte*> data has been written or <*min*> minutes have passed since the last checkpoint. Both arguments default to zero, in which case they are ignored. When the <*min*> argument is non-zero, an internal task will run every <*min*> minutes to perform the checkpoint. See the Berkeley DB reference guide for more details.

cryptfile <*file*>
Specify the pathname of a file containing an encryption key to use for encrypting the database. Encryption is performed using Berkeley DB's implementation of AES. Note that encryption can only be configured before any database files are created, and changing the key can only be done after destroying the current database and recreating it. Encryption is not enabled by default, and some distributions of Berkeley DB do not support encryption.

cryptkey <*key*>
Specify an encryption key to use for encrypting the database. This option may be used when a separate *cryptfile* is not desired. Only one of **cryptkey** or **cryptfile** may be configured.

dbconfig <*Berkeley−DB−setting*>
Specify a configuration directive to be placed in the **DB_CONFIG** file of the database directory. The **dbconfig** directive is just a convenience to allow all necessary configuration to be set in the **slapd.conf** file. The options set using this directive will only be written to the **DB_CONFIG** file if no such file existed at server startup time, otherwise they are completely ignored. This allows one to set initial values without overwriting/destroying a **DB_CONFIG** file that was already customized through other means. This directive may be specified multiple times, as needed. For example:

```
dbconfig set_cachesize 0 1048576 0
dbconfig set_lg_bsize 2097152
```

dbnosync
Specify that on-disk database contents should not be immediately synchronized with in memory changes. Enabling this option may improve performance at the expense of data security. See the Berkeley DB reference guide for more details.

directory <*directory*>
Specify the directory where the BDB files containing this database and associated indexes live. A separate directory must be specified for each database. The default is **/var/openldap/openldap-data**.

dirtyread
Allow reads of modified but not yet committed data. Usually transactions are isolated to prevent other operations from accessing uncommitted data. This option may improve performance, but may also return inconsistent results if the data comes from a transaction that is later aborted. In this case, the modified data is discarded and a subsequent search will return a different result.

dncachesize <*integer*>
Specify the maximum number of DNs in the in-memory DN cache. The default is twice the **cachesize**. Ideally this cache should be large enough to contain the DNs of every entry in the database.

idlcachesize <*integer*>
Specify the size of the in-memory index cache, in index slots. The default is zero. A larger value will speed up frequent searches of indexed entries. An **hdb** database needs a large **idlcachesize** for good search performance, typically three times the **cachesize** (entry cache size) or larger.

index {<*attrlist*>|**default**} [**pres**,**eq**,**approx**,**sub**,<*special*>]
Specify the indexes to maintain for the given attribute (or list of attributes). Some attributes only support a subset of indexes. If only an <*attr*> is given, the indices specified for **default** are maintained. Note that setting a default does not imply that all attributes will be indexed. Also, for best performance, an **eq** index should always be configured for the **objectClass** attribute.

A number of special index parameters may be specified. The index type **sub** can be decomposed into **subinitial**, **subany**, and **subfinal** indices. The special type **nolang** may be specified to disallow use of this index by language subtypes. The special type **nosubtypes** may be specified to disallow use of this index by named subtypes. Note: changing **index** settings in **slapd.conf**(5) requires rebuilding indices, see **slapindex**(8); changing **index** settings dynamically by LDAPModifying "cn=config" automatically causes rebuilding of the indices online in a background task.

linearindex
Tell **slapindex** to index one attribute at a time. By default, all indexed attributes in an entry are processed at the same time. With this option, each indexed attribute is processed individually, using multiple passes through the entire database. This option improves **slapindex** performance when the database size exceeds the **dbcache** size. When the **dbcache** is large enough, this option is not needed and will decrease performance. Also by default, **slapadd** performs full indexing and so a separate **slapindex** run is not needed. With this option, **slapadd** does no indexing and **slapindex** must be used.

lockdetect {**oldest**|**youngest**|**fewest**|**random**|**default**}
Specify which transaction to abort when a deadlock is detected. The default is **random**.

mode <*integer*>
Specify the file protection mode that newly created database index files should have. The default is 0600.

searchstack <*depth*>
Specify the depth of the stack used for search filter evaluation. Search filters are evaluated on a stack to accommodate nested AND / OR clauses. An individual stack is assigned to each server thread. The depth of the stack determines how complex a filter can be evaluated without requiring any additional memory allocation. Filters that are nested deeper than the search stack depth will cause a separate stack to be allocated for that particular search operation. These allocations can

have a major negative impact on server performance, but specifying too much stack will also consume a great deal of memory. Each search stack uses 512K bytes per level. The default stack depth is 16, thus 8MB per thread is used.

shm_key <*integer*>
Specify a key for a shared memory BDB environment. By default the BDB environment uses memory mapped files. If a non-zero value is specified, it will be used as the key to identify a shared memory region that will house the environment.

ACCESS CONTROL

The **bdb** and **hdb** backends honor access control semantics as indicated in **slapd.access**(5).

FILES

/etc/openldap/slapd.conf
default **slapd** configuration file

DB_CONFIG
Berkeley DB configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **slapadd**(8), **slapcat**(8), **slapindex**(8), Berkeley DB documentation.

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release. Originally begun by Kurt Zeilenga. Caching mechanisms originally designed by Jong-Hyuk Choi. Completion and subsequent work, as well as back-hdb, by Howard Chu.

NAME

slapd-config − configuration backend to slapd

SYNOPSIS

ETCDIR/slapd.d

DESCRIPTION

The **config** backend manages all of the configuration information for the **slapd**(8) daemon. This configuration information is also used by the SLAPD tools **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), and **slaptest**(8).

The **config** backend is backward compatible with the older **slapd.conf**(5) file but provides the ability to change the configuration dynamically at runtime. If slapd is run with only a **slapd.conf** file dynamic changes will be allowed but they will not persist across a server restart. Dynamic changes are only saved when slapd is running from a **slapd.d** configuration directory.

Unlike other backends, there can only be one instance of the **config** backend, and most of its structure is predefined. The root of the database is hardcoded to **cn=config** and this root entry contains global settings for slapd. Multiple child entries underneath the root entry are used to carry various other settings:

cn=Module
: dynamically loaded modules

cn=Schema
: schema definitions

olcBackend=xxx
: backend-specific settings

olcDatabase=xxx
: database-specific settings

The **cn=Module** entries will only appear in configurations where slapd was built with support for dynamically loaded modules. There can be multiple entries, one for each configured module path. Within each entry there will be values recorded for each module loaded on a given path. These entries have no children.

The **cn=Schema** entry contains all of the hardcoded schema elements. The children of this entry contain all user-defined schema elements. In schema that were loaded from include files, the child entry will be named after the include file from which the schema was loaded. Typically the first child in this subtree will be **cn=core,cn=schema,cn=config**.

olcBackend entries are for storing settings specific to a single backend type (and thus global to all database instances of that type). At present there are no backends that implement settings of this nature, so usually there will not be any olcBackend entries.

olcDatabase entries store settings specific to a single database instance. These entries may have **olcOverlay** child entries corresponding to any overlays configured on the database. The olcDatabase and olcOverlay entries may also have miscellaneous child entries for other settings as needed. There are two special database entries that are predefined - one is an entry for the config database itself, and the other is for the "frontend" database. Settings in the frontend database are inherited by the other databases, unless they are explicitly overridden in a specific database.

The specific configuration options available are discussed below in the Global Configuration Options, General Backend Options, and General Database Options. Options are set by defining LDAP attributes with specific values. In general the names of the LDAP attributes are the same as the corresponding **slapd.conf** keyword, with an "olc" prefix added on.

The parser for many of these attributes is the same as used for parsing the slapd.conf keywords. As such, slapd.conf keywords that allow multiple items to be specified on one line, separated by whitespace, will

allow multiple items to be specified in one attribute value. However, when reading the attribute via LDAP, the items will be returned as individual attribute values.

Backend-specific options are discussed in the **slapd-<backend>(5)** manual pages. Refer to the "OpenLDAP Administrator's Guide" for more details on configuring slapd.

GLOBAL CONFIGURATION OPTIONS

Options described in this section apply to the server as a whole. Arguments that should be replaced by actual text are shown in brackets <>.

These options may only be specified in the **cn=config** entry. This entry must have an objectClass of **olcGlobal**.

olcAllows: <features>

Specify a set of features to allow (default none). **bind_v2** allows acceptance of LDAPv2 bind requests. Note that **slapd**(8) does not truly implement LDAPv2 (RFC 1777), now Historic (RFC 3494). **bind_anon_cred** allows anonymous bind when credentials are not empty (e.g. when DN is empty). **bind_anon_dn** allows unauthenticated (anonymous) bind when DN is not empty. **update_anon** allows unauthenticated (anonymous) update operations to be processed (subject to access controls and other administrative limits). **proxy_authz_anon** allows unauthenticated (anonymous) proxy authorization control to be processed (subject to access controls, authorization and other administrative limits).

olcArgsFile: <filename>

The (absolute) name of a file that will hold the **slapd** server's command line options if started without the debugging command line option.

olcAttributeOptions: <option-name>...

Define tagging attribute options or option tag/range prefixes. Options must not end with '-', prefixes must end with '-'. The 'lang-' prefix is predefined. If you use the **olcAttributeOptions** directive, 'lang-' will no longer be defined and you must specify it explicitly if you want it defined.

An attribute description with a tagging option is a subtype of that attribute description without the option. Except for that, options defined this way have no special semantics. Prefixes defined this way work like the 'lang-' options: They define a prefix for tagging options starting with the prefix. That is, if you define the prefix 'x-foo-', you can use the option 'x-foo-bar'. Furthermore, in a search or compare, a prefix or range name (with a trailing '-') matches all options starting with that name, as well as the option with the range name sans the trailing '-'. That is, 'x-foo-bar-' matches 'x-foo-bar' and 'x-foo-bar-baz'.

RFC 4520 reserves options beginning with 'x-' for private experiments. Other options should be registered with IANA, see RFC 4520 section 3.5. OpenLDAP also has the 'binary' option built in, but this is a transfer option, not a tagging option.

olcAuthzPolicy: <policy>

Used to specify which rules to use for Proxy Authorization. Proxy authorization allows a client to authenticate to the server using one user's credentials, but specify a different identity to use for authorization and access control purposes. It essentially allows user A to login as user B, using user A's password. The **none** flag disables proxy authorization. This is the default setting. The **from** flag will use rules in the *authzFrom* attribute of the authorization DN. The **to** flag will use rules in the *authzTo* attribute of the authentication DN. The **any** flag, an alias for the deprecated value of **both**, will allow any of the above, whatever succeeds first (checked in **to**, **from** sequence. The **all** flag requires both authorizations to succeed.

The rules are mechanisms to specify which identities are allowed to perform proxy authorization. The *authzFrom* attribute in an entry specifies which other users are allowed to proxy login to this entry. The *authzTo* attribute in an entry specifies which other users this user can authorize as. Use

of *authzTo* rules can be easily abused if users are allowed to write arbitrary values to this attribute. In general the *authzTo* attribute must be protected with ACLs such that only privileged users can modify it. The value of *authzFrom* and *authzTo* describes an **identity** or a set of identities; it can take five forms:

ldap:///<base>??[<scope>]?<filter>
dn[.<dnstyle>]:<pattern>
u[<mech>[<realm>]]:<pattern>
group[/objectClass[/attributeType]]:<pattern>
<pattern>

<dnstyle>:={exact|onelevel|children|subtree|regex}

The first form is a valid LDAP **URI** where the <*host*>:<*port*>, the <*attrs*> and the <*extensions*> portions must be absent, so that the search occurs locally on either *authzFrom* or *authzTo*. The second form is a **DN**, with the optional style modifiers *exact*, *onelevel*, *children*, and *subtree* for exact, onelevel, children and subtree matches, which cause <*pattern*> to be normalized according to the DN normalization rules, or the special *regex* style, which causes the <*pattern*> to be treated as a POSIX (''extended'') regular expression, as discussed in **regex**(7) and/or **re_format**(7). A pattern of * means any non-anonymous DN. The third form is a SASL **id**, with the optional fields <*mech*> and <*realm*> that allow to specify a SASL **mechanism**, and eventually a SASL **realm**, for those mechanisms that support one. The need to allow the specification of a mechanism is still debated, and users are strongly discouraged to rely on this possibility. The fourth form is a group specification, consisting of the keyword **group**, optionally followed by the specification of the group **objectClass** and member **attributeType**. The group with DN **<pattern>** is searched with base scope, and in case of match, the values of the member **attributeType** are searched for the asserted DN. For backwards compatibility, if no identity type is provided, i.e. only **<pattern>** is present, an *exact DN* is assumed; as a consequence, **<pattern>** is subjected to DN normalization. Since the interpretation of *authzFrom* and *authzTo* can impact security, users are strongly encouraged to explicitly set the type of identity specification that is being used. A subset of these rules can be used as third arg in the **olcAuthzRegexp** statement (see below); significantly, the *URI* and the *dn.exact:*<*dn*> forms.

olcAuthzRegexp: <match> <replace>

Used by the authentication framework to convert simple user names, such as provided by SASL subsystem, to an LDAP DN used for authorization purposes. Note that the resultant DN need not refer to an existing entry to be considered valid. When an authorization request is received from the SASL subsystem, the SASL **USERNAME**, **REALM**, and **MECHANISM** are taken, when available, and combined into a name of the form

UID=<username>[[,CN=<realm>],CN=<mechanism>],CN=auth

This name is then compared against the **match** POSIX (''extended'') regular expression, and if the match is successful, the name is replaced with the **replace** string. If there are wildcard strings in the **match** regular expression that are enclosed in parenthesis, e.g.

UID=([^,]*),CN=.*

then the portion of the name that matched the wildcard will be stored in the numbered placeholder variable $1. If there are other wildcard strings in parenthesis, the matching strings will be in $2, $3, etc. up to $9. The placeholders can then be used in the **replace** string, e.g.

UID=$1,OU=Accounts,DC=example,DC=com

The replaced name can be either a DN, i.e. a string prefixed by "dn:", or an LDAP URI. If the latter, the server will use the URI to search its own database(s) and, if the search returns exactly one entry, the name is replaced by the DN of that entry. The LDAP URI must have no hostport, attrs,

or extensions components, but the filter is mandatory, e.g.

ldap:///OU=Accounts,DC=example,DC=com??one?(UID=$1)

The protocol portion of the URI must be strictly **ldap**. Note that this search is subject to access controls. Specifically, the authentication identity must have "auth" access in the subject.

Multiple **olcAuthzRegexp** values can be specified to allow for multiple matching and replacement patterns. The matching patterns are checked in the order they appear in the attribute, stopping at the first successful match.

olcConcurrency: <integer>
Specify a desired level of concurrency. Provided to the underlying thread system as a hint. The default is not to provide any hint. This setting is only meaningful on some platforms where there is not a one to one correspondence between user threads and kernel threads.

olcConnMaxPending: <integer>
Specify the maximum number of pending requests for an anonymous session. If requests are submitted faster than the server can process them, they will be queued up to this limit. If the limit is exceeded, the session is closed. The default is 100.

olcConnMaxPendingAuth: <integer>
Specify the maximum number of pending requests for an authenticated session. The default is 1000.

olcDisallows: <features>
Specify a set of features to disallow (default none). **bind_anon** disables acceptance of anonymous bind requests. Note that this setting does not prohibit anonymous directory access (See "require authc"). **bind_simple** disables simple (bind) authentication. **tls_2_anon** disables forcing session to anonymous status (see also **tls_authc**) upon StartTLS operation receipt. **tls_authc** disallows the StartTLS operation if authenticated (see also **tls_2_anon**).

olcGentleHUP: { TRUE | FALSE }
A SIGHUP signal will only cause a 'gentle' shutdown-attempt: **Slapd** will stop listening for new connections, but will not close the connections to the current clients. Future write operations return unwilling-to-perform, though. Slapd terminates when all clients have closed their connections (if they ever do), or − as before − if it receives a SIGTERM signal. This can be useful if you wish to terminate the server and start a new **slapd** server **with another database,** without disrupting the currently active clients. The default is FALSE. You may wish to use **olcIdleTimeout** along with this option.

olcIdleTimeout: <integer>
Specify the number of seconds to wait before forcibly closing an idle client connection. A setting of 0 disables this feature. The default is 0.

olcIndexIntLen: <integer>
Specify the key length for ordered integer indices. The most significant bytes of the binary integer will be used for index keys. The default value is 4, which provides exact indexing for 31 bit values. A floating point representation is used to index too large values.

olcIndexSubstrIfMaxlen: <integer>
Specify the maximum length for subinitial and subfinal indices. Only this many characters of an attribute value will be processed by the indexing functions; any excess characters are ignored. The default is 4.

olcIndexSubstrIfMinlen: <integer>
Specify the minimum length for subinitial and subfinal indices. An attribute value must have at least this many characters in order to be processed by the indexing functions. The default is 2.

olcIndexSubstrAnyLen: <integer>
Specify the length used for subany indices. An attribute value must have at least this many characters in order to be processed. Attribute values longer than this length will be processed in segments of this length. The default is 4. The subany index will also be used in subinitial and subfinal index lookups when the filter string is longer than the *olcIndexSubstrIfMaxlen* value.

olcIndexSubstrAnyStep: <integer>
Specify the steps used in subany index lookups. This value sets the offset for the segments of a filter string that are processed for a subany index lookup. The default is 2. For example, with the default values, a search using this filter "cn=*abcdefgh*" would generate index lookups for "abcd", "cdef", and "efgh".

Note: Indexing support depends on the particular backend in use. Also, changing these settings will generally require deleting any indices that depend on these parameters and recreating them with **slapindex**(8).

olcLocalSSF: <SSF>
Specifies the Security Strength Factor (SSF) to be given local LDAP sessions, such as those to the ldapi:// listener. For a description of SSF values, see **olcSaslSecProps**'s **minssf** option description. The default is 71.

olcLogFile: <filename>
Specify a file for recording debug log messages. By default these messages only go to stderr and are not recorded anywhere else. Specifying a logfile copies messages to both stderr and the logfile.

olcLogLevel: <integer> [...]
Specify the level at which debugging statements and operation statistics should be syslogged (currently logged to the **syslogd**(8) LOG_LOCAL4 facility). They must be considered subsystems rather than increasingly verbose log levels. Some messages with higher priority are logged regardless of the configured loglevel as soon as any logging is configured. Log levels are additive, and available levels are:

1	**(0x1 trace)** trace function calls
2	**(0x2 packets)** debug packet handling
4	**(0x4 args)** heavy trace debugging (function args)
8	**(0x8 conns)** connection management
16	**(0x10 BER)** print out packets sent and received
32	**(0x20 filter)** search filter processing
64	**(0x40 config)** configuration file processing
128	**(0x80 ACL)** access control list processing
256	**(0x100 stats)** stats log connections/operations/results
512	**(0x200 stats2)** stats log entries sent
1024	**(0x400 shell)** print communication with shell backends
2048	**(0x800 parse)** entry parsing
16384	**(0x4000 sync)** LDAPSync replication
32768	**(0x8000 none)** only messages that get logged whatever log level is set

The desired log level can be input as a single integer that combines the (ORed) desired levels, both in decimal or in hexadecimal notation, as a list of integers (that are ORed internally), or as a list of the names that are shown between brackets, such that

olcLogLevel: 129
olcLogLevel: 0x81
olcLogLevel: 128 1
olcLogLevel: 0x80 0x1
olcLogLevel: acl trace

are equivalent. The keyword **any** can be used as a shortcut to enable logging at all levels (equivalent to -1). The keyword **none**, or the equivalent integer representation, causes those messages that are logged regardless of the configured olcLogLevel to be logged. In fact, if no olcLogLevel (or a 0 level) is defined, no logging occurs, so at least the **none** level is required to have high priority messages logged.

olcPasswordCryptSaltFormat: <format>

Specify the format of the salt passed to **crypt**(3) when generating {CRYPT} passwords (see **olcPasswordHash**) during processing of LDAP Password Modify Extended Operations (RFC 3062).

This string needs to be in **sprintf**(3) format and may include one (and only one) %s conversion. This conversion will be substituted with a string of random characters from [A−Za−z0−9./]. For example, "%.2s" provides a two character salt and "1%.8s" tells some versions of crypt(3) to use an MD5 algorithm and provides 8 random characters of salt. The default is "%s", which provides 31 characters of salt.

olcPidFile: <filename>

The (absolute) name of a file that will hold the **slapd** server's process ID (see **getpid**(2)) if started without the debugging command line option.

olcPluginLogFile: <filename>

The (absolute) name of a file that will contain log messages from **SLAPI** plugins. See **slapd.plugin**(5) for details.

olcReferral: <url>

Specify the referral to pass back when **slapd**(8) cannot find a local database to handle a request. If multiple values are specified, each url is provided.

olcReverseLookup: TRUE | FALSE

Enable/disable client name unverified reverse lookup (default is **FALSE** if compiled with --enable-rlookups).

olcRootDSE: <file>

Specify the name of an LDIF(5) file containing user defined attributes for the root DSE. These attributes are returned in addition to the attributes normally produced by slapd.

The root DSE is an entry with information about the server and its capabilities, in operational attributes. It has the empty DN, and can be read with e.g.:

ldapsearch -x -b "" -s base "+"

See RFC 4512 section 5.1 for details.

olcSaslHost: <fqdn>

Used to specify the fully qualified domain name used for SASL processing.

olcSaslRealm: <realm>

Specify SASL realm. Default is empty.

olcSaslSecProps: <properties>

Used to specify Cyrus SASL security properties. The **none** flag (without any other properties) causes the flag properties default, "noanonymous,noplain", to be cleared. The **noplain** flag disables mechanisms susceptible to simple passive attacks. The **noactive** flag disables mechanisms susceptible to active attacks. The **nodict** flag disables mechanisms susceptible to passive dictionary attacks. The **noanonymous** flag disables mechanisms which support anonymous login. The

forwardsec flag require forward secrecy between sessions. The **passcred** require mechanisms which pass client credentials (and allow mechanisms which can pass credentials to do so). The **minssf=<factor>** property specifies the minimum acceptable *security strength factor* as an integer approximate to effective key length used for encryption. 0 (zero) implies no protection, 1 implies integrity protection only, 56 allows DES or other weak ciphers, 112 allows triple DES and other strong ciphers, 128 allows RC4, Blowfish and other modern strong ciphers. The default is 0. The **maxssf=<factor>** property specifies the maximum acceptable *security strength factor* as an integer (see minssf description). The default is INT_MAX. The **maxbufsize=<size>** property specifies the maximum security layer receive buffer size allowed. 0 disables security layers. The default is 65536.

olcServerID: <integer> [<URL>]
Specify an integer ID from 0 to 4095 for this server. These IDs are required when using multimaster replication and each master must have a unique ID. If the URL is provided, this directive may be specified multiple times, providing a complete list of participating servers and their IDs. The fully qualified hostname of each server should be used in the supplied URLs. The IDs are used in the "replica id" field of all CSNs generated by the specified server. The default value is zero. Example:

olcServerID: 1 ldap://ldap1.example.com
olcServerID: 2 ldap://ldap2.example.com

olcSockbufMaxIncoming: <integer>
Specify the maximum incoming LDAP PDU size for anonymous sessions. The default is 262143.

olcSockbufMaxIncomingAuth: <integer>
Specify the maximum incoming LDAP PDU size for authenticated sessions. The default is 4194303.

olcThreads: <integer>
Specify the maximum size of the primary thread pool. The default is 16; the minimum value is 2.

olcToolThreads: <integer>
Specify the maximum number of threads to use in tool mode. This should not be greater than the number of CPUs in the system. The default is 1.

TLS OPTIONS

If **slapd** is built with support for Transport Layer Security, there are more options you can specify.

olcTLSCipherSuite: <cipher-suite-spec>
Permits configuring what ciphers will be accepted and the preference order. <cipher-suite-spec> should be a cipher specification for OpenSSL. Example:

olcTLSCipherSuite: HIGH:MEDIUM:+SSLv2

To check what ciphers a given spec selects in OpenSSL, use:

openssl ciphers -v <cipher-suite-spec>

To obtain the list of ciphers in GNUtls use:

gnutls-cli -l

olcTLSCACertificateFile: <filename>
Specifies the file that contains certificates for all of the Certificate Authorities that **slapd** will recognize.

olcTLSCACertificatePath: <path>
Specifies the path of a directory that contains Certificate Authority certificates in separate individual files. Usually only one of this or the olcTLSCACertificateFile is defined. If both are specified,

both locations will be used. This directive is not supported when using GNUtls.

olcTLSCertificateFile: <filename>
Specifies the file that contains the **slapd** server certificate.

olcTLSCertificateKeyFile: <filename>
Specifies the file that contains the **slapd** server private key that matches the certificate stored in the **olcTLSCertificateFile** file. If the private key is protected with a password, the password must be manually typed in when slapd starts. Usually the private key is not protected with a password, to allow slapd to start without manual intervention, so it is of critical importance that the file is protected carefully.

olcTLSDHParamFile: <filename>
This directive specifies the file that contains parameters for Diffie-Hellman ephemeral key exchange. This is required in order to use a DSA certificate on the server. If multiple sets of parameters are present in the file, all of them will be processed. Note that setting this option may also enable Anonymous Diffie-Hellman key exchanges in certain non-default cipher suites. You should append "!ADH" to your cipher suites if you have changed them from the default, otherwise no certificate exchanges or verification will be done. When using GNUtls these parameters are always generated randomly so this directive is ignored.

olcTLSRandFile: <filename>
Specifies the file to obtain random bits from when /dev/[u]random is not available. Generally set to the name of the EGD/PRNGD socket. The environment variable RANDFILE can also be used to specify the filename. This directive is ignored with GNUtls.

olcTLSVerifyClient: <level>
Specifies what checks to perform on client certificates in an incoming TLS session, if any. The **<level>** can be specified as one of the following keywords:

never This is the default. **slapd** will not ask the client for a certificate.

allow The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, it will be ignored and the session proceeds normally.

try The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, the session is immediately terminated.

demand | hard | true
These keywords are all equivalent, for compatibility reasons. The client certificate is requested. If no certificate is provided, or a bad certificate is provided, the session is immediately terminated.

Note that a valid client certificate is required in order to use the SASL EXTERNAL authentication mechanism with a TLS session. As such, a non-default **olcTLSVerifyClient** setting must be chosen to enable SASL EXTERNAL authentication.

olcTLSCRLCheck: <level>
Specifies if the Certificate Revocation List (CRL) of the CA should be used to verify if the client certificates have not been revoked. This requires **olcTLSCACertificatePath** parameter to be set. This parameter is ignored with GNUtls. **<level>** can be specified as one of the following keywords:

none No CRL checks are performed

peer Check the CRL of the peer certificate

all Check the CRL for a whole certificate chain

olcTLSCRLFile: <filename>
Specifies a file containing a Certificate Revocation List to be used for verifying that certificates have not been revoked. This parameter is only valid when using GNUtls.

DYNAMIC MODULE OPTIONS

If **slapd** is compiled with --enable-modules then the module-related entries will be available. These entries are named **cn=module{x},cn=config** and must have the olcModuleList objectClass. One entry should be created per **olcModulePath.** Normally the config engine generates the "{x}" index in the RDN automatically, so it can be omitted when initially loading these entries.

olcModuleLoad: <filename>
Specify the name of a dynamically loadable module to load. The filename may be an absolute path name or a simple filename. Non-absolute names are searched for in the directories specified by the **olcModulePath** option.

olcModulePath: <pathspec>
Specify a list of directories to search for loadable modules. Typically the path is colon-separated but this depends on the operating system.

SCHEMA OPTIONS

Schema definitions are created as entries in the **cn=schema,cn=config** subtree. These entries must have the olcSchemaConfig objectClass. As noted above, the actual **cn=schema,cn=config** entry is predefined and any values specified for it are ignored.

olcAttributetypes: (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oid>] [EQUALITY <oid>] [ORDERING <oid>] [SUBSTR <oid>] [SYNTAX <oidlen>] [SINGLE−VALUE] [COLLECTIVE] [NO−USER−MODIFICATION] [USAGE <attributeUsage>])
Specify an attribute type using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **olcObjectIdentifier** description.)

olcDitContentRules: (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [AUX <oids>] [MUST <oids>] [MAY <oids>] [NOT <oids>])
Specify an DIT Content Rule using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **olcObjectIdentifier** description.)

olcObjectClasses: (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oids>] [{ ABSTRACT | STRUCTURAL | AUXILIARY }] [MUST <oids>] [MAY <oids>])
Specify an objectclass using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the object class OID. (See the **olcObjectIdentifier** description.) Object classes are "STRUCTURAL" by default.

olcObjectIdentifier: <name> { <oid> | <name>[:<suffix>] }
Define a string name that equates to the given OID. The string can be used in place of the numeric OID in objectclass and attribute definitions. The name can also be used with a suffix of the form ":xx" in which case the value "oid.xx" will be used.

GENERAL BACKEND OPTIONS

Options in these entries only apply to the configuration of a single type of backend. All backends may support this class of options. The entry must be named **olcBackend=<databasetype>,cn=config** and must have the olcBackendConfig objectClass. <databasetype> should be one of **bdb**, **config**, **dnssrv**, **hdb**, **ldap**, **ldif**, **meta**, **monitor**, **null**, **passwd**, **perl**, **relay**, **shell**, or **sql**. At present, no backend implements any options of this type.

DATABASE OPTIONS

Database options are set in entries named **olcDatabase={x}<databasetype>,cn=config** and must have the olcDatabaseConfig objectClass. Normally the config engine generates the "{x}" index in the RDN automatically, so it can be omitted when initially loading these entries.

The special frontend database is always numbered "{-1}" and the config database is always numbered "{0}".

GLOBAL DATABASE OPTIONS

Options in this section may be set in the special "frontend" database and inherited in all the other databases. These options may be altered by further settings in each specific database. The frontend entry must be named **olcDatabase=frontend,cn=config** and must have the olcFrontendConfig objectClass.

olcAccess: to <what> [by <who> <access> <control>]+

Grant access (specified by <access>) to a set of entries and/or attributes (specified by <what>) by one or more requestors (specified by <who>). If no access controls are present, the default policy allows anyone and everyone to read anything but restricts updates to rootdn. (e.g., "olcAccess: to * by * read"). See **slapd.access**(5) and the "OpenLDAP Administrator's Guide" for details.

Access controls set in the frontend are appended to any access controls set on the specific databases. The rootdn of a database can always read and write EVERYTHING in that database.

Extra special care must be taken with the access controls on the config database. Unlike other databases, the default policy for the config database is to only allow access to the rootdn. Regular users should not have read access, and write access should be granted very carefully to privileged administrators.

olcDefaultSearchBase: <dn>

Specify a default search base to use when client submits a non-base search request with an empty base DN. Base scoped search requests with an empty base DN are not affected. This setting is only allowed in the frontend entry.

olcPasswordHash: <hash> [<hash>...]

This option configures one or more hashes to be used in generation of user passwords stored in the userPassword attribute during processing of LDAP Password Modify Extended Operations (RFC 3062). The <hash> must be one of **{SSHA}**, **{SHA}**, **{SMD5}**, **{MD5}**, **{CRYPT}**, and **{CLEARTEXT}**. The default is **{SSHA}**.

{SHA} and **{SSHA}** use the SHA-1 algorithm (FIPS 160-1), the latter with a seed.

{MD5} and **{SMD5}** use the MD5 algorithm (RFC 1321), the latter with a seed.

{CRYPT} uses the **crypt**(3).

{CLEARTEXT} indicates that the new password should be added to userPassword as clear text.

Note that this option does not alter the normal user applications handling of userPassword during LDAP Add, Modify, or other LDAP operations. This setting is only allowed in the frontend entry.

olcReadOnly: TRUE | FALSE

This option puts the database into "read-only" mode. Any attempts to modify the database will return an "unwilling to perform" error. By default, olcReadOnly is FALSE. Note that when this option is set TRUE on the frontend, it cannot be reset without restarting the server, since further writes to the config database will be rejected.

olcRequires: <conditions>

Specify a set of conditions to require (default none). The directive may be specified globally and/or per-database; databases inherit global conditions, so per-database specifications are additive. **bind** requires bind operation prior to directory operations. **LDAPv3** requires session to be using LDAP version 3. **authc** requires authentication prior to directory operations. **SASL** requires SASL authentication prior to directory operations. **strong** requires strong authentication prior to directory operations. The strong keyword allows protected "simple" authentication as well as SASL authentication. **none** may be used to require no conditions (useful to clear out globally set conditions within a particular database); it must occur first in the list of conditions.

olcRestrict: <oplist>

Specify a list of operations that are restricted. Restrictions on a specific database override any frontend setting. Operations can be any of **add**, **bind**, **compare**, **delete**, **extended[=<OID>]**, **modify**, **rename**, **search**, or the special pseudo-operations **read** and **write**, which respectively summarize read and write operations. The use of *restrict write* is equivalent to *olcReadOnly: TRUE* (see above). The **extended** keyword allows to indicate the OID of the specific operation to be restricted.

olcSchemaDN: <dn>

Specify the distinguished name for the subschema subentry that controls the entries on this server. The default is "cn=Subschema".

olcSecurity: <factors>

Specify a set of security strength factors (separated by white space) to require (see **olcSaslSecprops**'s **minssf** option for a description of security strength factors). The directive may be specified globally and/or per-database. **ssf=<n>** specifies the overall security strength factor. **transport=<n>** specifies the transport security strength factor. **tls=<n>** specifies the TLS security strength factor. **sasl=<n>** specifies the SASL security strength factor. **update_ssf=<n>** specifies the overall security strength factor to require for directory updates. **update_transport=<n>** specifies the transport security strength factor to require for directory updates. **update_tls=<n>** specifies the TLS security strength factor to require for directory updates. **update_sasl=<n>** specifies the SASL security strength factor to require for directory updates. **simple_bind=<n>** specifies the security strength factor required for *simple* username/password authentication. Note that the **transport** factor is measure of security provided by the underlying transport, e.g. ldapi:// (and eventually IPSEC). It is not normally used.

olcSizeLimit: {<integer>|unlimited}

olcSizeLimit: size[.{soft|hard|unchecked}]=<integer> [...]

Specify the maximum number of entries to return from a search operation. The default size limit is 500. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the size limits. Extra args can be added in the same value or as additional values. See **olcLimits** for an explanation of the different flags.

olcSortVals <attr> [...]

Specify a list of multi-valued attributes whose values will always be maintained in sorted order. Using this option will allow Modify, Compare, and filter evaluations on these attributes to be performed more efficiently. The resulting sort order depends on the attributes' syntax and matching rules and may not correspond to lexical order or any other recognizable order. This setting is only allowed in the frontend entry.

olcTimeLimit: {<integer>|unlimited}

olcTimeLimit: time[.{soft|hard}]=<integer> [...]

Specify the maximum number of seconds (in real time) **slapd** will spend answering a search request. The default time limit is 3600. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the time limits. Extra args can be added in the same value or as additional values. See **olcLimits** for an explanation of the different flags.

GENERAL DATABASE OPTIONS

Options in this section only apply to the specific database for which they are defined. They are supported by every type of backend. All of the Global Database Options may also be used here.

olcHidden: TRUE | FALSE

Controls whether the database will be used to answer queries. A database that is hidden will never be selected to answer any queries, and any suffix configured on the database will be ignored in checks for conflicts with other databases. By default, olcHidden is FALSE.

olcLastMod: TRUE | FALSE

Controls whether **slapd** will automatically maintain the modifiersName, modifyTimestamp, creatorsName, and createTimestamp attributes for entries. It also controls the entryCSN and entryUUID attributes, which are needed by the syncrepl provider. By default, olcLastMod is TRUE.

olcLimits: <who> <limit> [<limit> [...]]

Specify time and size limits based on who initiated an operation. The argument **who** can be any of

anonymous | users | [dn[.<style>]=]<pattern> | group[/oc[/at]]=<pattern>

with

<style> ::= exact | base | onelevel | subtree | children | regex | anonymous

The term **anonymous** matches all unauthenticated clients. The term **users** matches all authenticated clients; otherwise an **exact** dn pattern is assumed unless otherwise specified by qualifying the (optional) key string **dn** with **exact** or **base** (which are synonyms), to require an exact match; with **onelevel**, to require exactly one level of depth match; with **subtree**, to allow any level of depth match, including the exact match; with **children**, to allow any level of depth match, not including the exact match; **regex** explicitly requires the (default) match based on POSIX (''extended'') regular expression pattern. Finally, **anonymous** matches unbound operations; the **pattern** field is ignored. The same behavior is obtained by using the **anonymous** form of the **who** clause. The term **group**, with the optional objectClass **oc** and attributeType **at** fields, followed by **pattern**, sets the limits for any DN listed in the values of the **at** attribute (default **member**) of the **oc** group objectClass (default **groupOfNames**) whose DN exactly matches **pattern**.

The currently supported limits are **size** and **time**.

The syntax for time limits is **time[.{soft|hard}]=<integer>**, where *integer* is the number of seconds slapd will spend answering a search request. If no time limit is explicitly requested by the client, the **soft** limit is used; if the requested time limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for time limits smaller or equal to the **hard** limit are honored. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

The syntax for size limits is **size[.{soft|hard|unchecked}]=<integer>**, where *integer* is the maximum number of entries slapd will return answering a search request. If no size limit is explicitly requested by the client, the **soft** limit is used; if the requested size limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for size limits smaller or equal to the **hard** limit are honored. The **unchecked** specifier sets a limit on the number of candidates a search request is allowed to examine. The rationale behind it is that searches for non-properly indexed attributes may result in large sets of candidates, which must be examined by **slapd**(8) to determine whether they match the search filter or not. The **unchecked** limit provides a means to drop such operations before they are even started. If the selected candidates exceed the **unchecked** limit, the search will abort with *Unwilling to perform*.

If it is set to the keyword *unlimited*, no limit is applied (the default). If it is set to *disable*, the search is not even performed; this can be used to disallow searches for a specific set of users. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

In case of no match, the global limits are used. The default values are the same as **olcSizeLimit** and **olcTimeLimit**; no limit is set on **unchecked**.

If **pagedResults** control is requested, the **hard** size limit is used by default, because the request of a specific page size is considered an explicit request for a limitation on the number of entries to be returned. However, the size limit applies to the total count of entries returned within the search, and not to a single page. Additional size limits may be enforced; the syntax is **size.pr={<integer>|noEstimate|unlimited}**, where *integer* is the max page size if no explicit limit is set; the keyword *noEstimate* inhibits the server from returning an estimate of the total number of entries that might be returned (note: the current implementation does not return any estimate). The keyword *unlimited* indicates that no limit is applied to the pagedResults control page size. The syntax **size.prtotal={<integer>|unlimited|disabled}** allows to set a limit on the total number of entries that a pagedResults control allows to return. By default it is set to the **hard** limit. When set, *integer* is the max number of entries that the whole search with pagedResults control can return. Use *unlimited* to allow unlimited number of entries to be returned, e.g. to allow the use of the pagedResults control as a means to circumvent size limitations on regular searches; the keyword *disabled* disables the control, i.e. no paged results can be returned. Note that the total number of entries returned when the pagedResults control is requested cannot exceed the **hard** size limit of regular searches unless extended by the **prtotal** switch.

olcMaxDerefDepth: <depth>

Specifies the maximum number of aliases to dereference when trying to resolve an entry, used to avoid infinite alias loops. The default is 15.

olcMirrorMode: TRUE | FALSE

This option puts a replica database into "mirror" mode. Update operations will be accepted from any user, not just the updatedn. The database must already be configured as syncrepl consumer before this keyword may be set. This mode also requires a **olcServerID** (see above) to be configured. By default, this setting is FALSE.

olcPlugin: <plugin_type> <lib_path> <init_function> [<arguments>]

Configure a SLAPI plugin. See the **slapd.plugin**(5) manpage for more details.

olcRootDN: <dn>

Specify the distinguished name that is not subject to access control or administrative limit restrictions for operations on this database. This DN may or may not be associated with an entry. An empty root DN (the default) specifies no root access is to be granted. It is recommended that the rootdn only be specified when needed (such as when initially populating a database). If the rootdn is within a namingContext (suffix) of the database, a simple bind password may also be provided using the **olcRootPW** directive. Note that the rootdn is always needed when using syncrepl.

olcRootPW: <password>

Specify a password (or hash of the password) for the rootdn. The password can only be set if the rootdn is within the namingContext (suffix) of the database. This option accepts all RFC 2307 userPassword formats known to the server (see **olcPasswordHash** description) as well as cleartext. **slappasswd**(8) may be used to generate a hash of a password. Cleartext and **{CRYPT}** passwords are not recommended. If empty (the default), authentication of the root DN is by other means (e.g. SASL). Use of SASL is encouraged.

olcSubordinate: [TRUE | FALSE | advertise]

Specify that the current backend database is a subordinate of another backend database. A subordinate database may have only one suffix. This option may be used to glue multiple

databases into a single namingContext. If the suffix of the current database is within the namingContext of a superior database, searches against the superior database will be propagated to the subordinate as well. All of the databases associated with a single namingContext should have identical rootdns. Behavior of other LDAP operations is unaffected by this setting. In particular, it is not possible to use moddn to move an entry from one subordinate to another subordinate within the namingContext.

If the optional **advertise** flag is supplied, the naming context of this database is advertised in the root DSE. The default is to hide this database context, so that only the superior context is visible.

If the slap tools **slapcat**(8), **slapadd**(8), or **slapindex**(8) are used on the superior database, any glued subordinates that support these tools are opened as well.

Databases that are glued together should usually be configured with the same indices (assuming they support indexing), even for attributes that only exist in some of these databases. In general, all of the glued databases should be configured as similarly as possible, since the intent is to provide the appearance of a single directory.

Note that the subordinate functionality is implemented internally by the *glue* overlay and as such its behavior will interact with other overlays in use. By default, the glue overlay is automatically configured as the last overlay on the superior database. Its position on the database can be explicitly configured by setting an **overlay glue** directive at the desired position. This explicit configuration is necessary e.g. when using the *syncprov* overlay, which needs to follow *glue* in order to work over all of the glued databases. E.g.

```
dn: olcDatabase={1}bdb,cn=config
olcSuffix: dc=example,dc=com
...

dn: olcOverlay={0}glue,olcDatabase={1}bdb,cn=config
...

dn: olcOverlay={1}syncprov,olcDatabase={1}bdb,cn=config
...
```

See the Overlays section below for more details.

olcSuffix: <dn suffix>

Specify the DN suffix of queries that will be passed to this backend database. Multiple suffix lines can be given and at least one is required for each database definition. If the suffix of one database is "inside" that of another, the database with the inner suffix must come first in the configuration file.

olcSyncrepl: rid=<replica ID> provider=ldap[s]://<hostname>[:port] searchbase=<base DN> [type=refreshOnly|refreshAndPersist] [interval=dd:hh:mm:ss] [retry=[<retry interval> <# of retries>]+] [filter=<filter str>] [scope=sub|one|base|subord] [attrs=<attr list>] [exattrs=<attr list>] [attrsonly] [sizelimit=<limit>] [timelimit=<limit>] [schemachecking=on|off] [bindmethod=simple|sasl] [binddn=<dn>] [saslmech=<mech>] [authcid=<identity>] [authzid=<identity>] [credentials=<passwd>] [realm=<realm>] [secprops=<properties>] [starttls=yes|critical] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all] [logbase=<base DN>] [logfilter=<filter str>] [syncdata=default|accesslog|changelog]

Specify the current database as a replica which is kept up-to-date with the master content by establishing the current **slapd**(8) as a replication consumer site running a **syncrepl** replication engine. The replica content is kept synchronized to the master content using the LDAP Content Synchronization protocol. Refer to the "OpenLDAP Administrator's Guide" for detailed information on setting up a replicated **slapd** directory service using the **syncrepl** replication

engine.

rid identifies the current **syncrepl** directive within the replication consumer site. It is a non-negative integer having no more than three digits.

provider specifies the replication provider site containing the master content as an LDAP URI. If <port> is not given, the standard LDAP port number (389 or 636) is used.

The content of the **syncrepl** replica is defined using a search specification as its result set. The consumer **slapd** will send search requests to the provider **slapd** according to the search specification. The search specification includes **searchbase, scope, filter, attrs, attrsonly, sizelimit,** and **timelimit** parameters as in the normal search specification. The **exattrs** option may also be used to specify attributes that should be omitted from incoming entries. The **scope** defaults to **sub**, the **filter** defaults to **(objectclass=*)**, and there is no default **searchbase**. The **attrs** list defaults to **"*,+"** to return all user and operational attributes, and **attrsonly** and **exattrs** are unset by default. The **sizelimit** and **timelimit** only accept "unlimited" and positive integers, and both default to "unlimited". Note, however, that any provider-side limits for the replication identity will be enforced by the provider regardless of the limits requested by the LDAP Content Synchronization operation, much like for any other search operation.

The LDAP Content Synchronization protocol has two operation types. In the **refreshOnly** operation, the next synchronization search operation is periodically rescheduled at an interval time (specified by **interval** parameter; 1 day by default) after each synchronization operation finishes. In the **refreshAndPersist** operation, a synchronization search remains persistent in the provider slapd. Further updates to the master replica will generate **searchResultEntry** to the consumer slapd as the search responses to the persistent synchronization search.

If an error occurs during replication, the consumer will attempt to reconnect according to the **retry** parameter which is a list of the <retry interval> and <# of retries> pairs. For example, retry="60 10 300 3" lets the consumer retry every 60 seconds for the first 10 times and then retry every 300 seconds for the next 3 times before stop retrying. The ‘+’ in <# of retries> means indefinite number of retries until success.

The schema checking can be enforced at the LDAP Sync consumer site by turning on the **schemachecking** parameter. The default is off.

A **bindmethod** of **simple** requires the options **binddn** and **credentials** and should only be used when adequate security services (e.g. TLS or IPSEC) are in place. A **bindmethod** of **sasl** requires the option **saslmech.** Depending on the mechanism, an authentication identity and/or credentials can be specified using **authcid** and **credentials.** The **authzid** parameter may be used to specify an authorization identity. Specific security properties (as with the **sasl-secprops** keyword above) for a SASL bind can be set with the **secprops** option. A non default SASL realm can be set with the **realm** option. The provider, other than allow authentication of the syncrepl identity, should grant that identity appropriate access privileges to the data that is being replicated (**access** directive), and appropriate time and size limits (**limits** directive).

The **starttls** parameter specifies use of the StartTLS extended operation to establish a TLS session before Binding to the provider. If the **critical** argument is supplied, the session will be aborted if the StartTLS request fails. Otherwise the syncrepl session continues without TLS. The tls_reqcert setting defaults to "demand" and the other TLS settings default to the same as the main slapd TLS settings.

Rather than replicating whole entries, the consumer can query logs of data modifications. This

mode of operation is referred to as *delta syncrepl*. In addition to the above parameters, the **logbase** and **logfilter** parameters must be set appropriately for the log that will be used. The **syncdata** parameter must be set to either "accesslog" if the log conforms to the **slapo-accesslog**(5) log format, or "changelog" if the log conforms to the obsolete *changelog* format. If the **syncdata** parameter is omitted or set to "default" then the log parameters are ignored.

olcUpdateDN: <dn>
This option is only applicable in a slave database. It specifies the DN permitted to update (subject to access controls) the replica. It is only needed in certain push-mode replication scenarios. Generally, this DN *should not* be the same as the **rootdn** used at the master.

olcUpdateRef: <url>
Specify the referral to pass back when **slapd**(8) is asked to modify a replicated local database. If multiple values are specified, each url is provided.

DATABASE-SPECIFIC OPTIONS

Each database may allow specific configuration options; they are documented separately in the backends' manual pages. See the **slapd.backends**(5) manual page for an overview of available backends.

OVERLAYS

An overlay is a piece of code that intercepts database operations in order to extend or change them. Overlays are pushed onto a stack over the database, and so they will execute in the reverse of the order in which they were configured and the database itself will receive control last of all.

Overlays must be configured as child entries of a specific database. The entry's RDN must be of the form **olcOverlay={x}<overlaytype>** and the entry must have the olcOverlayConfig objectClass. Normally the config engine generates the "{x}" index in the RDN automatically, so it can be omitted when initially loading these entries.

See the **slapd.overlays**(5) manual page for an overview of available overlays.

EXAMPLES

Here is a short example of a configuration in LDIF suitable for use with **slapadd**(8) :

```
dn: cn=config
objectClass: olcGlobal
cn: config
olcPidFile: LOCALSTATEDIR/run/slapd.pid
olcAttributeOptions: x-hidden lang-

dn: cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema

include: SYSCONFDIR/schema/core.ldif

dn: olcDatabase=frontend,cn=config
objectClass: olcDatabaseConfig
objectClass: olcFrontendConfig
olcDatabase: frontend
# Subtypes of "name" (e.g. "cn" and "ou") with the
# option ";x-hidden" can be searched for/compared,
# but are not shown.  See slapd.access(5).
olcAccess: to attrs=name;x-hidden by * =cs
# Protect passwords.  See slapd.access(5).
olcAccess: to attrs=userPassword  by * auth
# Read access to other attributes and entries.
```

```
olcAccess: to * by * read

# set a rootpw for the config database so we can bind.
# deny access to everyone else.
dn: olcDatabase=config,cn=config
objectClass: olcDatabaseConfig
olcDatabase: config
olcRootPW: {SSHA}XKYnrjvGT3wZFQrDD5040US592LxsdLy
olcAccess: to * by * none

dn: olcDatabase=bdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcBdbConfig
olcDatabase: bdb
olcSuffix: "dc=our-domain,dc=com"
# The database directory MUST exist prior to
# running slapd AND should only be accessible
# by the slapd/tools. Mode 0700 recommended.
olcDbDirectory: LOCALSTATEDIR/openldap-data
# Indices to maintain
olcDbIndex:     objectClass  eq
olcDbIndex:     cn,sn,mail   pres,eq,approx,sub

# We serve small clients that do not handle referrals,
# so handle remote lookups on their behalf.
dn: olcDatabase=ldap,cn=config
objectClass: olcDatabaseConfig
objectClass: olcLdapConfig
olcDatabase: ldap
olcSuffix: ""
olcDbUri: ldap://ldap.some-server.com/
```

Assuming the above data was saved in a file named "config.ldif" and the ETCDIR/slapd.d directory has been created, this command will initialize the configuration:

```
slapadd -F ETCDIR/slapd.d -n 0 -l config.ldif
```

"OpenLDAP Administrator’s Guide" contains a longer annotated example of a slapd configuration.

Alternatively, an existing slapd.conf file can be converted to the new format using slapd or any of the slap tools:

```
slaptest -f ETCDIR/slapd.conf -F ETCDIR/slapd.d
```

FILES

ETCDIR/slapd.conf
: default slapd configuration file

ETCDIR/slapd.d
: default slapd configuration directory

SEE ALSO

ldap(3), **ldif**(5), **slapd.access**(5), **slapd.backends**(5), **slapd.conf**(5), **slapd.overlays**(5), **slapd.plugin**(5), **slapd.replog**(5), **slapd**(8), **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), **slappasswd**(8), **slaptest**(8).

"OpenLDAP Administrator’s Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

NAME

slapd-config − configuration backend to slapd

SYNOPSIS

/etc/openldap/slapd.d

DESCRIPTION

The **config** backend manages all of the configuration information for the **slapd**(8) daemon. This configuration information is also used by the SLAPD tools **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), and **slaptest**(8).

The **config** backend is backward compatible with the older **slapd.conf**(5) file but provides the ability to change the configuration dynamically at runtime. If slapd is run with only a **slapd.conf** file dynamic changes will be allowed but they will not persist across a server restart. Dynamic changes are only saved when slapd is running from a **slapd.d** configuration directory.

Unlike other backends, there can only be one instance of the **config** backend, and most of its structure is predefined. The root of the database is hardcoded to **cn=config** and this root entry contains global settings for slapd. Multiple child entries underneath the root entry are used to carry various other settings:

cn=Module
: dynamically loaded modules

cn=Schema
: schema definitions

olcBackend=xxx
: backend-specific settings

olcDatabase=xxx
: database-specific settings

The **cn=Module** entries will only appear in configurations where slapd was built with support for dynamically loaded modules. There can be multiple entries, one for each configured module path. Within each entry there will be values recorded for each module loaded on a given path. These entries have no children.

The **cn=Schema** entry contains all of the hardcoded schema elements. The children of this entry contain all user-defined schema elements. In schema that were loaded from include files, the child entry will be named after the include file from which the schema was loaded. Typically the first child in this subtree will be **cn=core,cn=schema,cn=config**.

olcBackend entries are for storing settings specific to a single backend type (and thus global to all database instances of that type). At present there are no backends that implement settings of this nature, so usually there will not be any olcBackend entries.

olcDatabase entries store settings specific to a single database instance. These entries may have **olcOverlay** child entries corresponding to any overlays configured on the database. The olcDatabase and olcOverlay entries may also have miscellaneous child entries for other settings as needed. There are two special database entries that are predefined - one is an entry for the config database itself, and the other is for the "frontend" database. Settings in the frontend database are inherited by the other databases, unless they are explicitly overridden in a specific database.

The specific configuration options available are discussed below in the Global Configuration Options, General Backend Options, and General Database Options. Options are set by defining LDAP attributes with specific values. In general the names of the LDAP attributes are the same as the corresponding **slapd.conf** keyword, with an "olc" prefix added on.

The parser for many of these attributes is the same as used for parsing the slapd.conf keywords. As such, slapd.conf keywords that allow multiple items to be specified on one line, separated by whitespace, will

allow multiple items to be specified in one attribute value. However, when reading the attribute via LDAP, the items will be returned as individual attribute values.

Backend-specific options are discussed in the **slapd-<backend>(5)** manual pages. Refer to the "OpenLDAP Administrator's Guide" for more details on configuring slapd.

GLOBAL CONFIGURATION OPTIONS

Options described in this section apply to the server as a whole. Arguments that should be replaced by actual text are shown in brackets <>.

These options may only be specified in the **cn=config** entry. This entry must have an objectClass of **olcGlobal**.

olcAllows: <features>

Specify a set of features to allow (default none). **bind_v2** allows acceptance of LDAPv2 bind requests. Note that **slapd**(8) does not truly implement LDAPv2 (RFC 1777), now Historic (RFC 3494). **bind_anon_cred** allows anonymous bind when credentials are not empty (e.g. when DN is empty). **bind_anon_dn** allows unauthenticated (anonymous) bind when DN is not empty. **update_anon** allows unauthenticated (anonymous) update operations to be processed (subject to access controls and other administrative limits). **proxy_authz_anon** allows unauthenticated (anonymous) proxy authorization control to be processed (subject to access controls, authorization and other administrative limits).

olcArgsFile: <filename>

The (absolute) name of a file that will hold the **slapd** server's command line options if started without the debugging command line option.

olcAttributeOptions: <option-name>...

Define tagging attribute options or option tag/range prefixes. Options must not end with '-', prefixes must end with '-'. The 'lang-' prefix is predefined. If you use the **olcAttributeOptions** directive, 'lang-' will no longer be defined and you must specify it explicitly if you want it defined.

An attribute description with a tagging option is a subtype of that attribute description without the option. Except for that, options defined this way have no special semantics. Prefixes defined this way work like the 'lang-' options: They define a prefix for tagging options starting with the prefix. That is, if you define the prefix 'x-foo-', you can use the option 'x-foo-bar'. Furthermore, in a search or compare, a prefix or range name (with a trailing '-') matches all options starting with that name, as well as the option with the range name sans the trailing '-'. That is, 'x-foo-bar-' matches 'x-foo-bar' and 'x-foo-bar-baz'.

RFC 4520 reserves options beginning with 'x-' for private experiments. Other options should be registered with IANA, see RFC 4520 section 3.5. OpenLDAP also has the 'binary' option built in, but this is a transfer option, not a tagging option.

olcAuthzPolicy: <policy>

Used to specify which rules to use for Proxy Authorization. Proxy authorization allows a client to authenticate to the server using one user's credentials, but specify a different identity to use for authorization and access control purposes. It essentially allows user A to login as user B, using user A's password. The **none** flag disables proxy authorization. This is the default setting. The **from** flag will use rules in the *authzFrom* attribute of the authorization DN. The **to** flag will use rules in the *authzTo* attribute of the authentication DN. The **any** flag, an alias for the deprecated value of **both**, will allow any of the above, whatever succeeds first (checked in **to**, **from** sequence. The **all** flag requires both authorizations to succeed.

The rules are mechanisms to specify which identities are allowed to perform proxy authorization. The *authzFrom* attribute in an entry specifies which other users are allowed to proxy login to this entry. The *authzTo* attribute in an entry specifies which other users this user can authorize as. Use

of *authzTo* rules can be easily abused if users are allowed to write arbitrary values to this attribute. In general the *authzTo* attribute must be protected with ACLs such that only privileged users can modify it. The value of *authzFrom* and *authzTo* describes an **identity** or a set of identities; it can take five forms:

ldap:///<base>??[<scope>]?<filter>
dn[.<dnstyle>]:<pattern>
u[<mech>[<realm>]]:<pattern>
group[/objectClass[/attributeType]]:<pattern>
<pattern>

<dnstyle>:={exact|onelevel|children|subtree|regex}

The first form is a valid LDAP **URI** where the <*host*>:<*port*>, the <*attrs*> and the <*extensions*> portions must be absent, so that the search occurs locally on either *authzFrom* or *authzTo*. The second form is a **DN**, with the optional style modifiers *exact*, *onelevel*, *children*, and *subtree* for exact, onelevel, children and subtree matches, which cause <*pattern*> to be normalized according to the DN normalization rules, or the special *regex* style, which causes the <*pattern*> to be treated as a POSIX (''extended'') regular expression, as discussed in **regex**(7) and/or **re_format**(7). A pattern of * means any non-anonymous DN. The third form is a SASL **id**, with the optional fields <*mech*> and <*realm*> that allow to specify a SASL **mechanism**, and eventually a SASL **realm**, for those mechanisms that support one. The need to allow the specification of a mechanism is still debated, and users are strongly discouraged to rely on this possibility. The fourth form is a group specification, consisting of the keyword **group**, optionally followed by the specification of the group **objectClass** and member **attributeType**. The group with DN **<pattern>** is searched with base scope, and in case of match, the values of the member **attributeType** are searched for the asserted DN. For backwards compatibility, if no identity type is provided, i.e. only **<pattern>** is present, an *exact DN* is assumed; as a consequence, **<pattern>** is subjected to DN normalization. Since the interpretation of *authzFrom* and *authzTo* can impact security, users are strongly encouraged to explicitly set the type of identity specification that is being used. A subset of these rules can be used as third arg in the **olcAuthzRegexp** statement (see below); significantly, the *URI* and the *dn.exact:*<*dn*> forms.

olcAuthzRegexp: <match> <replace>

Used by the authentication framework to convert simple user names, such as provided by SASL subsystem, to an LDAP DN used for authorization purposes. Note that the resultant DN need not refer to an existing entry to be considered valid. When an authorization request is received from the SASL subsystem, the SASL **USERNAME**, **REALM**, and **MECHANISM** are taken, when available, and combined into a name of the form

UID=<username>[[,CN=<realm>],CN=<mechanism>],CN=auth

This name is then compared against the **match** POSIX (''extended'') regular expression, and if the match is successful, the name is replaced with the **replace** string. If there are wildcard strings in the **match** regular expression that are enclosed in parenthesis, e.g.

UID=([^,]*),CN=.*

then the portion of the name that matched the wildcard will be stored in the numbered placeholder variable $1. If there are other wildcard strings in parenthesis, the matching strings will be in $2, $3, etc. up to $9. The placeholders can then be used in the **replace** string, e.g.

UID=$1,OU=Accounts,DC=example,DC=com

The replaced name can be either a DN, i.e. a string prefixed by "dn:", or an LDAP URI. If the latter, the server will use the URI to search its own database(s) and, if the search returns exactly one entry, the name is replaced by the DN of that entry. The LDAP URI must have no hostport, attrs,

or extensions components, but the filter is mandatory, e.g.

ldap:///OU=Accounts,DC=example,DC=com??one?(UID=$1)

The protocol portion of the URI must be strictly **ldap**. Note that this search is subject to access controls. Specifically, the authentication identity must have "auth" access in the subject.

Multiple **olcAuthzRegexp** values can be specified to allow for multiple matching and replacement patterns. The matching patterns are checked in the order they appear in the attribute, stopping at the first successful match.

olcConcurrency: <integer>

Specify a desired level of concurrency. Provided to the underlying thread system as a hint. The default is not to provide any hint. This setting is only meaningful on some platforms where there is not a one to one correspondence between user threads and kernel threads.

olcConnMaxPending: <integer>

Specify the maximum number of pending requests for an anonymous session. If requests are submitted faster than the server can process them, they will be queued up to this limit. If the limit is exceeded, the session is closed. The default is 100.

olcConnMaxPendingAuth: <integer>

Specify the maximum number of pending requests for an authenticated session. The default is 1000.

olcDisallows: <features>

Specify a set of features to disallow (default none). **bind_anon** disables acceptance of anonymous bind requests. Note that this setting does not prohibit anonymous directory access (See "require authc"). **bind_simple** disables simple (bind) authentication. **tls_2_anon** disables forcing session to anonymous status (see also **tls_authc**) upon StartTLS operation receipt. **tls_authc** disallows the StartTLS operation if authenticated (see also **tls_2_anon**).

olcGentleHUP: { TRUE | FALSE }

A SIGHUP signal will only cause a 'gentle' shutdown-attempt: **Slapd** will stop listening for new connections, but will not close the connections to the current clients. Future write operations return unwilling-to-perform, though. Slapd terminates when all clients have closed their connections (if they ever do), or − as before − if it receives a SIGTERM signal. This can be useful if you wish to terminate the server and start a new **slapd** server **with another database,** without disrupting the currently active clients. The default is FALSE. You may wish to use **olcIdleTimeout** along with this option.

olcIdleTimeout: <integer>

Specify the number of seconds to wait before forcibly closing an idle client connection. A setting of 0 disables this feature. The default is 0.

olcIndexIntLen: <integer>

Specify the key length for ordered integer indices. The most significant bytes of the binary integer will be used for index keys. The default value is 4, which provides exact indexing for 31 bit values. A floating point representation is used to index too large values.

olcIndexSubstrIfMaxlen: <integer>

Specify the maximum length for subinitial and subfinal indices. Only this many characters of an attribute value will be processed by the indexing functions; any excess characters are ignored. The default is 4.

olcIndexSubstrIfMinlen: <integer>

Specify the minimum length for subinitial and subfinal indices. An attribute value must have at least this many characters in order to be processed by the indexing functions. The default is 2.

olcIndexSubstrAnyLen: <integer>

Specify the length used for subany indices. An attribute value must have at least this many characters in order to be processed. Attribute values longer than this length will be processed in segments of this length. The default is 4. The subany index will also be used in subinitial and subfinal index lookups when the filter string is longer than the *olcIndexSubstrIfMaxlen* value.

olcIndexSubstrAnyStep: <integer>

Specify the steps used in subany index lookups. This value sets the offset for the segments of a filter string that are processed for a subany index lookup. The default is 2. For example, with the default values, a search using this filter "cn=*abcdefgh*" would generate index lookups for "abcd", "cdef", and "efgh".

Note: Indexing support depends on the particular backend in use. Also, changing these settings will generally require deleting any indices that depend on these parameters and recreating them with **slapindex**(8).

olcLocalSSF: <SSF>

Specifies the Security Strength Factor (SSF) to be given local LDAP sessions, such as those to the ldapi:// listener. For a description of SSF values, see **olcSaslSecProps**'s **minssf** option description. The default is 71.

olcLogFile: <filename>

Specify a file for recording debug log messages. By default these messages only go to stderr and are not recorded anywhere else. Specifying a logfile copies messages to both stderr and the logfile.

olcLogLevel: <integer> [...]

Specify the level at which debugging statements and operation statistics should be syslogged (currently logged to the **syslogd**(8) LOG_LOCAL4 facility). They must be considered subsystems rather than increasingly verbose log levels. Some messages with higher priority are logged regardless of the configured loglevel as soon as any logging is configured. Log levels are additive, and available levels are:

1	**(0x1 trace)** trace function calls
2	**(0x2 packets)** debug packet handling
4	**(0x4 args)** heavy trace debugging (function args)
8	**(0x8 conns)** connection management
16	**(0x10 BER)** print out packets sent and received
32	**(0x20 filter)** search filter processing
64	**(0x40 config)** configuration file processing
128	**(0x80 ACL)** access control list processing
256	**(0x100 stats)** stats log connections/operations/results
512	**(0x200 stats2)** stats log entries sent
1024	**(0x400 shell)** print communication with shell backends
2048	**(0x800 parse)** entry parsing
16384	**(0x4000 sync)** LDAPSync replication
32768	**(0x8000 none)** only messages that get logged whatever log level is set

The desired log level can be input as a single integer that combines the (ORed) desired levels, both in decimal or in hexadecimal notation, as a list of integers (that are ORed internally), or as a list of the names that are shown between brackets, such that

olcLogLevel: 129
olcLogLevel: 0x81
olcLogLevel: 128 1
olcLogLevel: 0x80 0x1
olcLogLevel: acl trace

are equivalent. The keyword **any** can be used as a shortcut to enable logging at all levels (equivalent to -1). The keyword **none**, or the equivalent integer representation, causes those messages that are logged regardless of the configured olcLogLevel to be logged. In fact, if no olcLogLevel (or a 0 level) is defined, no logging occurs, so at least the **none** level is required to have high priority messages logged.

olcPasswordCryptSaltFormat: <format>

Specify the format of the salt passed to **crypt**(3) when generating {CRYPT} passwords (see **olcPasswordHash**) during processing of LDAP Password Modify Extended Operations (RFC 3062).

This string needs to be in **sprintf**(3) format and may include one (and only one) %s conversion. This conversion will be substituted with a string of random characters from [A−Za−z0−9./]. For example, "%.2s" provides a two character salt and "1%.8s" tells some versions of crypt(3) to use an MD5 algorithm and provides 8 random characters of salt. The default is "%s", which provides 31 characters of salt.

olcPidFile: <filename>

The (absolute) name of a file that will hold the **slapd** server's process ID (see **getpid**(2)) if started without the debugging command line option.

olcPluginLogFile: <filename>

The (absolute) name of a file that will contain log messages from **SLAPI** plugins. See **slapd.plugin**(5) for details.

olcReferral: <url>

Specify the referral to pass back when **slapd**(8) cannot find a local database to handle a request. If multiple values are specified, each url is provided.

olcReverseLookup: TRUE | FALSE

Enable/disable client name unverified reverse lookup (default is **FALSE** if compiled with --enable-rlookups).

olcRootDSE: <file>

Specify the name of an LDIF(5) file containing user defined attributes for the root DSE. These attributes are returned in addition to the attributes normally produced by slapd.

The root DSE is an entry with information about the server and its capabilities, in operational attributes. It has the empty DN, and can be read with e.g.:

ldapsearch -x -b "" -s base "+"

See RFC 4512 section 5.1 for details.

olcSaslHost: <fqdn>

Used to specify the fully qualified domain name used for SASL processing.

olcSaslRealm: <realm>

Specify SASL realm. Default is empty.

olcSaslSecProps: <properties>

Used to specify Cyrus SASL security properties. The **none** flag (without any other properties) causes the flag properties default, "noanonymous,noplain", to be cleared. The **noplain** flag disables mechanisms susceptible to simple passive attacks. The **noactive** flag disables mechanisms susceptible to active attacks. The **nodict** flag disables mechanisms susceptible to passive dictionary attacks. The **noanonymous** flag disables mechanisms which support anonymous login. The

forwardsec flag require forward secrecy between sessions. The **passcred** require mechanisms which pass client credentials (and allow mechanisms which can pass credentials to do so). The **minssf=<factor>** property specifies the minimum acceptable *security strength factor* as an integer approximate to effective key length used for encryption. 0 (zero) implies no protection, 1 implies integrity protection only, 56 allows DES or other weak ciphers, 112 allows triple DES and other strong ciphers, 128 allows RC4, Blowfish and other modern strong ciphers. The default is 0. The **maxssf=<factor>** property specifies the maximum acceptable *security strength factor* as an integer (see minssf description). The default is INT_MAX. The **maxbufsize=<size>** property specifies the maximum security layer receive buffer size allowed. 0 disables security layers. The default is 65536.

olcServerID: <integer> [<URL>]

Specify an integer ID from 0 to 4095 for this server. These IDs are required when using multimaster replication and each master must have a unique ID. If the URL is provided, this directive may be specified multiple times, providing a complete list of participating servers and their IDs. The fully qualified hostname of each server should be used in the supplied URLs. The IDs are used in the "replica id" field of all CSNs generated by the specified server. The default value is zero. Example:

olcServerID: 1 ldap://ldap1.example.com
olcServerID: 2 ldap://ldap2.example.com

olcSockbufMaxIncoming: <integer>

Specify the maximum incoming LDAP PDU size for anonymous sessions. The default is 262143.

olcSockbufMaxIncomingAuth: <integer>

Specify the maximum incoming LDAP PDU size for authenticated sessions. The default is 4194303.

olcThreads: <integer>

Specify the maximum size of the primary thread pool. The default is 16; the minimum value is 2.

olcToolThreads: <integer>

Specify the maximum number of threads to use in tool mode. This should not be greater than the number of CPUs in the system. The default is 1.

TLS OPTIONS

If **slapd** is built with support for Transport Layer Security, there are more options you can specify.

olcTLSCipherSuite: <cipher-suite-spec>

Permits configuring what ciphers will be accepted and the preference order. <cipher-suite-spec> should be a cipher specification for OpenSSL. Example:

olcTLSCipherSuite: HIGH:MEDIUM:+SSLv2

To check what ciphers a given spec selects in OpenSSL, use:

openssl ciphers -v <cipher-suite-spec>

To obtain the list of ciphers in GNUtls use:

gnutls-cli -l

olcTLSCACertificateFile: <filename>

Specifies the file that contains certificates for all of the Certificate Authorities that **slapd** will recognize.

olcTLSCACertificatePath: <path>

Specifies the path of a directory that contains Certificate Authority certificates in separate individual files. Usually only one of this or the olcTLSCACertificateFile is defined. If both are specified,

both locations will be used. This directive is not supported when using GNUtls.

olcTLSCertificateFile: <filename>
Specifies the file that contains the **slapd** server certificate.

olcTLSCertificateKeyFile: <filename>
Specifies the file that contains the **slapd** server private key that matches the certificate stored in the **olcTLSCertificateFile** file. If the private key is protected with a password, the password must be manually typed in when slapd starts. Usually the private key is not protected with a password, to allow slapd to start without manual intervention, so it is of critical importance that the file is protected carefully.

olcTLSDHParamFile: <filename>
This directive specifies the file that contains parameters for Diffie-Hellman ephemeral key exchange. This is required in order to use a DSA certificate on the server. If multiple sets of parameters are present in the file, all of them will be processed. Note that setting this option may also enable Anonymous Diffie-Hellman key exchanges in certain non-default cipher suites. You should append "!ADH" to your cipher suites if you have changed them from the default, otherwise no certificate exchanges or verification will be done. When using GNUtls these parameters are always generated randomly so this directive is ignored.

olcTLSRandFile: <filename>
Specifies the file to obtain random bits from when /dev/[u]random is not available. Generally set to the name of the EGD/PRNGD socket. The environment variable RANDFILE can also be used to specify the filename. This directive is ignored with GNUtls.

olcTLSVerifyClient: <level>
Specifies what checks to perform on client certificates in an incoming TLS session, if any. The **<level>** can be specified as one of the following keywords:

never This is the default. **slapd** will not ask the client for a certificate.

allow The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, it will be ignored and the session proceeds normally.

try The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, the session is immediately terminated.

demand | hard | true
These keywords are all equivalent, for compatibility reasons. The client certificate is requested. If no certificate is provided, or a bad certificate is provided, the session is immediately terminated.

Note that a valid client certificate is required in order to use the SASL EXTERNAL authentication mechanism with a TLS session. As such, a non-default **olcTLSVerifyClient** setting must be chosen to enable SASL EXTERNAL authentication.

olcTLSCRLCheck: <level>
Specifies if the Certificate Revocation List (CRL) of the CA should be used to verify if the client certificates have not been revoked. This requires **olcTLSCACertificatePath** parameter to be set. This parameter is ignored with GNUtls. **<level>** can be specified as one of the following keywords:

none No CRL checks are performed

peer Check the CRL of the peer certificate

all Check the CRL for a whole certificate chain

olcTLSCRLFile: <filename>
Specifies a file containing a Certificate Revocation List to be used for verifying that certificates have not been revoked. This parameter is only valid when using GNUtls.

DYNAMIC MODULE OPTIONS

If **slapd** is compiled with --enable-modules then the module-related entries will be available. These entries are named **cn=module{x},cn=config** and must have the olcModuleList objectClass. One entry should be created per **olcModulePath.** Normally the config engine generates the "{x}" index in the RDN automatically, so it can be omitted when initially loading these entries.

olcModuleLoad: <filename>
Specify the name of a dynamically loadable module to load. The filename may be an absolute path name or a simple filename. Non-absolute names are searched for in the directories specified by the **olcModulePath** option.

olcModulePath: <pathspec>
Specify a list of directories to search for loadable modules. Typically the path is colon-separated but this depends on the operating system.

SCHEMA OPTIONS

Schema definitions are created as entries in the **cn=schema,cn=config** subtree. These entries must have the olcSchemaConfig objectClass. As noted above, the actual **cn=schema,cn=config** entry is predefined and any values specified for it are ignored.

olcAttributetypes: (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oid>] [EQUALITY <oid>] [ORDERING <oid>] [SUBSTR <oid>] [SYNTAX <oidlen>] [SINGLE−VALUE] [COLLECTIVE] [NO−USER−MODIFICATION] [USAGE <attributeUsage>])
Specify an attribute type using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **olcObjectIdentifier** description.)

olcDitContentRules: (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [AUX <oids>] [MUST <oids>] [MAY <oids>] [NOT <oids>])
Specify an DIT Content Rule using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **olcObjectIdentifier** description.)

olcObjectClasses: (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oids>] [{ ABSTRACT | STRUCTURAL | AUXILIARY }] [MUST <oids>] [MAY <oids>])
Specify an objectclass using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the object class OID. (See the **olcObjectIdentifier** description.) Object classes are "STRUCTURAL" by default.

olcObjectIdentifier: <name> { <oid> | <name>[:<suffix>] }
Define a string name that equates to the given OID. The string can be used in place of the numeric OID in objectclass and attribute definitions. The name can also be used with a suffix of the form ":xx" in which case the value "oid.xx" will be used.

GENERAL BACKEND OPTIONS

Options in these entries only apply to the configuration of a single type of backend. All backends may support this class of options. The entry must be named **olcBackend=<databasetype>,cn=config** and must have the olcBackendConfig objectClass. <databasetype> should be one of **bdb**, **config**, **dnssrv**, **hdb**, **ldap**, **ldif**, **meta**, **monitor**, **null**, **passwd**, **perl**, **relay**, **shell**, or **sql**. At present, no backend implements any options of this type.

DATABASE OPTIONS

Database options are set in entries named **olcDatabase={x}<databasetype>,cn=config** and must have the olcDatabaseConfig objectClass. Normally the config engine generates the "{x}" index in the RDN automatically, so it can be omitted when initially loading these entries.

The special frontend database is always numbered "{-1}" and the config database is always numbered "{0}".

GLOBAL DATABASE OPTIONS

Options in this section may be set in the special "frontend" database and inherited in all the other databases. These options may be altered by further settings in each specific database. The frontend entry must be named **olcDatabase=frontend,cn=config** and must have the olcFrontendConfig objectClass.

olcAccess: to <what> [by <who> <access> <control>]+

Grant access (specified by <access>) to a set of entries and/or attributes (specified by <what>) by one or more requestors (specified by <who>). If no access controls are present, the default policy allows anyone and everyone to read anything but restricts updates to rootdn. (e.g., "olcAccess: to * by * read"). See **slapd.access**(5) and the "OpenLDAP Administrator's Guide" for details.

Access controls set in the frontend are appended to any access controls set on the specific databases. The rootdn of a database can always read and write EVERYTHING in that database.

Extra special care must be taken with the access controls on the config database. Unlike other databases, the default policy for the config database is to only allow access to the rootdn. Regular users should not have read access, and write access should be granted very carefully to privileged administrators.

olcDefaultSearchBase: <dn>

Specify a default search base to use when client submits a non-base search request with an empty base DN. Base scoped search requests with an empty base DN are not affected. This setting is only allowed in the frontend entry.

olcPasswordHash: <hash> [<hash>...]

This option configures one or more hashes to be used in generation of user passwords stored in the userPassword attribute during processing of LDAP Password Modify Extended Operations (RFC 3062). The <hash> must be one of **{SSHA}**, **{SHA}**, **{SMD5}**, **{MD5}**, **{CRYPT}**, and **{CLEARTEXT}**. The default is **{SSHA}**.

{SHA} and **{SSHA}** use the SHA-1 algorithm (FIPS 160-1), the latter with a seed.

{MD5} and **{SMD5}** use the MD5 algorithm (RFC 1321), the latter with a seed.

{CRYPT} uses the **crypt**(3).

{CLEARTEXT} indicates that the new password should be added to userPassword as clear text.

Note that this option does not alter the normal user applications handling of userPassword during LDAP Add, Modify, or other LDAP operations. This setting is only allowed in the frontend entry.

olcReadOnly: TRUE | FALSE

This option puts the database into "read-only" mode. Any attempts to modify the database will return an "unwilling to perform" error. By default, olcReadOnly is FALSE. Note that when this option is set TRUE on the frontend, it cannot be reset without restarting the server, since further writes to the config database will be rejected.

olcRequires: <conditions>

Specify a set of conditions to require (default none). The directive may be specified globally and/or per-database; databases inherit global conditions, so per-database specifications are additive. **bind** requires bind operation prior to directory operations. **LDAPv3** requires session to be using LDAP version 3. **authc** requires authentication prior to directory operations. **SASL** requires SASL authentication prior to directory operations. **strong** requires strong authentication prior to directory operations. The strong keyword allows protected "simple" authentication as well as SASL authentication. **none** may be used to require no conditions (useful to clear out globally set conditions within a particular database); it must occur first in the list of conditions.

olcRestrict: <oplist>

Specify a list of operations that are restricted. Restrictions on a specific database override any frontend setting. Operations can be any of **add**, **bind**, **compare**, **delete**, **extended[=<OID>]**, **modify**, **rename**, **search**, or the special pseudo-operations **read** and **write**, which respectively summarize read and write operations. The use of *restrict write* is equivalent to *olcReadOnly: TRUE* (see above). The **extended** keyword allows to indicate the OID of the specific operation to be restricted.

olcSchemaDN: <dn>

Specify the distinguished name for the subschema subentry that controls the entries on this server. The default is "cn=Subschema".

olcSecurity: <factors>

Specify a set of security strength factors (separated by white space) to require (see **olcSaslSecprops**'s **minssf** option for a description of security strength factors). The directive may be specified globally and/or per-database. **ssf=<n>** specifies the overall security strength factor. **transport=<n>** specifies the transport security strength factor. **tls=<n>** specifies the TLS security strength factor. **sasl=<n>** specifies the SASL security strength factor. **update_ssf=<n>** specifies the overall security strength factor to require for directory updates. **update_transport=<n>** specifies the transport security strength factor to require for directory updates. **update_tls=<n>** specifies the TLS security strength factor to require for directory updates. **update_sasl=<n>** specifies the SASL security strength factor to require for directory updates. **simple_bind=<n>** specifies the security strength factor required for *simple* username/password authentication. Note that the **transport** factor is measure of security provided by the underlying transport, e.g. ldapi:// (and eventually IPSEC). It is not normally used.

olcSizeLimit: {<integer>|unlimited}

olcSizeLimit: size[.{soft|hard|unchecked}]=<integer> [...]

Specify the maximum number of entries to return from a search operation. The default size limit is 500. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the size limits. Extra args can be added in the same value or as additional values. See **olcLimits** for an explanation of the different flags.

olcSortVals <attr> [...]

Specify a list of multi-valued attributes whose values will always be maintained in sorted order. Using this option will allow Modify, Compare, and filter evaluations on these attributes to be performed more efficiently. The resulting sort order depends on the attributes' syntax and matching rules and may not correspond to lexical order or any other recognizable order. This setting is only allowed in the frontend entry.

olcTimeLimit: {<integer>|unlimited}

olcTimeLimit: time[.{soft|hard}]=<integer> [...]

Specify the maximum number of seconds (in real time) **slapd** will spend answering a search request. The default time limit is 3600. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the time limits. Extra args can be added in the same value or as additional values. See **olcLimits** for an explanation of the different flags.

GENERAL DATABASE OPTIONS

Options in this section only apply to the specific database for which they are defined. They are supported by every type of backend. All of the Global Database Options may also be used here.

olcHidden: TRUE | FALSE

Controls whether the database will be used to answer queries. A database that is hidden will never be selected to answer any queries, and any suffix configured on the database will be ignored in checks for conflicts with other databases. By default, olcHidden is FALSE.

olcLastMod: TRUE | FALSE

Controls whether **slapd** will automatically maintain the modifiersName, modifyTimestamp, creatorsName, and createTimestamp attributes for entries. It also controls the entryCSN and entryUUID attributes, which are needed by the syncrepl provider. By default, olcLastMod is TRUE.

olcLimits: <who> <limit> [<limit> [...]]

Specify time and size limits based on who initiated an operation. The argument **who** can be any of

anonymous | users | [dn[.<style>]=]<pattern> | group[/oc[/at]]=<pattern>

with

<style> ::= exact | base | onelevel | subtree | children | regex | anonymous

The term **anonymous** matches all unauthenticated clients. The term **users** matches all authenticated clients; otherwise an **exact** dn pattern is assumed unless otherwise specified by qualifying the (optional) key string **dn** with **exact** or **base** (which are synonyms), to require an exact match; with **onelevel**, to require exactly one level of depth match; with **subtree**, to allow any level of depth match, including the exact match; with **children**, to allow any level of depth match, not including the exact match; **regex** explicitly requires the (default) match based on POSIX (''extended'') regular expression pattern. Finally, **anonymous** matches unbound operations; the **pattern** field is ignored. The same behavior is obtained by using the **anonymous** form of the **who** clause. The term **group**, with the optional objectClass **oc** and attributeType **at** fields, followed by **pattern**, sets the limits for any DN listed in the values of the **at** attribute (default **member**) of the **oc** group objectClass (default **groupOfNames**) whose DN exactly matches **pattern**.

The currently supported limits are **size** and **time**.

The syntax for time limits is **time[.{soft|hard}]=<integer>**, where *integer* is the number of seconds slapd will spend answering a search request. If no time limit is explicitly requested by the client, the **soft** limit is used; if the requested time limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for time limits smaller or equal to the **hard** limit are honored. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

The syntax for size limits is **size[.{soft|hard|unchecked}]=<integer>**, where *integer* is the maximum number of entries slapd will return answering a search request. If no size limit is explicitly requested by the client, the **soft** limit is used; if the requested size limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for size limits smaller or equal to the **hard** limit are honored. The **unchecked** specifier sets a limit on the number of candidates a search request is allowed to examine. The rationale behind it is that searches for non-properly indexed attributes may result in large sets of candidates, which must be examined by **slapd**(8) to determine whether they match the search filter or not. The **unchecked** limit provides a means to drop such operations before they are even started. If the selected candidates exceed the **unchecked** limit, the search will abort with *Unwilling to perform*.

If it is set to the keyword *unlimited*, no limit is applied (the default). If it is set to *disable*, the search is not even performed; this can be used to disallow searches for a specific set of users. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

In case of no match, the global limits are used. The default values are the same as **olcSizeLimit** and **olcTimeLimit**; no limit is set on **unchecked**.

If **pagedResults** control is requested, the **hard** size limit is used by default, because the request of a specific page size is considered an explicit request for a limitation on the number of entries to be returned. However, the size limit applies to the total count of entries returned within the search, and not to a single page. Additional size limits may be enforced; the syntax is **size.pr={<integer>|noEstimate|unlimited}**, where *integer* is the max page size if no explicit limit is set; the keyword *noEstimate* inhibits the server from returning an estimate of the total number of entries that might be returned (note: the current implementation does not return any estimate). The keyword *unlimited* indicates that no limit is applied to the pagedResults control page size. The syntax **size.prtotal={<integer>|unlimited|disabled}** allows to set a limit on the total number of entries that a pagedResults control allows to return. By default it is set to the **hard** limit. When set, *integer* is the max number of entries that the whole search with pagedResults control can return. Use *unlimited* to allow unlimited number of entries to be returned, e.g. to allow the use of the pagedResults control as a means to circumvent size limitations on regular searches; the keyword *disabled* disables the control, i.e. no paged results can be returned. Note that the total number of entries returned when the pagedResults control is requested cannot exceed the **hard** size limit of regular searches unless extended by the **prtotal** switch.

olcMaxDerefDepth: <depth>

Specifies the maximum number of aliases to dereference when trying to resolve an entry, used to avoid infinite alias loops. The default is 15.

olcMirrorMode: TRUE | FALSE

This option puts a replica database into "mirror" mode. Update operations will be accepted from any user, not just the updatedn. The database must already be configured as syncrepl consumer before this keyword may be set. This mode also requires a **olcServerID** (see above) to be configured. By default, this setting is FALSE.

olcPlugin: <plugin_type> <lib_path> <init_function> [<arguments>]

Configure a SLAPI plugin. See the **slapd.plugin**(5) manpage for more details.

olcRootDN: <dn>

Specify the distinguished name that is not subject to access control or administrative limit restrictions for operations on this database. This DN may or may not be associated with an entry. An empty root DN (the default) specifies no root access is to be granted. It is recommended that the rootdn only be specified when needed (such as when initially populating a database). If the rootdn is within a namingContext (suffix) of the database, a simple bind password may also be provided using the **olcRootPW** directive. Note that the rootdn is always needed when using syncrepl.

olcRootPW: <password>

Specify a password (or hash of the password) for the rootdn. The password can only be set if the rootdn is within the namingContext (suffix) of the database. This option accepts all RFC 2307 userPassword formats known to the server (see **olcPasswordHash** description) as well as cleartext. **slappasswd**(8) may be used to generate a hash of a password. Cleartext and **{CRYPT}** passwords are not recommended. If empty (the default), authentication of the root DN is by other means (e.g. SASL). Use of SASL is encouraged.

olcSubordinate: [TRUE | FALSE | advertise]

Specify that the current backend database is a subordinate of another backend database. A subordinate database may have only one suffix. This option may be used to glue multiple

databases into a single namingContext. If the suffix of the current database is within the namingContext of a superior database, searches against the superior database will be propagated to the subordinate as well. All of the databases associated with a single namingContext should have identical rootdns. Behavior of other LDAP operations is unaffected by this setting. In particular, it is not possible to use moddn to move an entry from one subordinate to another subordinate within the namingContext.

If the optional **advertise** flag is supplied, the naming context of this database is advertised in the root DSE. The default is to hide this database context, so that only the superior context is visible.

If the slap tools **slapcat**(8), **slapadd**(8), or **slapindex**(8) are used on the superior database, any glued subordinates that support these tools are opened as well.

Databases that are glued together should usually be configured with the same indices (assuming they support indexing), even for attributes that only exist in some of these databases. In general, all of the glued databases should be configured as similarly as possible, since the intent is to provide the appearance of a single directory.

Note that the subordinate functionality is implemented internally by the *glue* overlay and as such its behavior will interact with other overlays in use. By default, the glue overlay is automatically configured as the last overlay on the superior database. Its position on the database can be explicitly configured by setting an **overlay glue** directive at the desired position. This explicit configuration is necessary e.g. when using the *syncprov* overlay, which needs to follow *glue* in order to work over all of the glued databases. E.g.

```
dn: olcDatabase={1}bdb,cn=config
olcSuffix: dc=example,dc=com
...

dn: olcOverlay={0}glue,olcDatabase={1}bdb,cn=config
...

dn: olcOverlay={1}syncprov,olcDatabase={1}bdb,cn=config
...
```

See the Overlays section below for more details.

olcSuffix: <dn suffix>

Specify the DN suffix of queries that will be passed to this backend database. Multiple suffix lines can be given and at least one is required for each database definition. If the suffix of one database is "inside" that of another, the database with the inner suffix must come first in the configuration file.

olcSyncrepl: rid=<replica ID> provider=ldap[s]://<hostname>[:port] searchbase=<base DN> [type=refreshOnly|refreshAndPersist] [interval=dd:hh:mm:ss] [retry=[<retry interval> <# of retries>]+] [filter=<filter str>] [scope=sub|one|base|subord] [attrs=<attr list>] [exattrs=<attr list>] [attrsonly] [sizelimit=<limit>] [timelimit=<limit>] [schemachecking=on|off] [bindmethod=simple|sasl] [binddn=<dn>] [saslmech=<mech>] [authcid=<identity>] [authzid=<identity>] [credentials=<passwd>] [realm=<realm>] [secprops=<properties>] [starttls=yes|critical] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all] [logbase=<base DN>] [logfilter=<filter str>] [syncdata=default|accesslog|changelog]

Specify the current database as a replica which is kept up-to-date with the master content by establishing the current **slapd**(8) as a replication consumer site running a **syncrepl** replication engine. The replica content is kept synchronized to the master content using the LDAP Content Synchronization protocol. Refer to the "OpenLDAP Administrator's Guide" for detailed information on setting up a replicated **slapd** directory service using the **syncrepl** replication

engine.

rid identifies the current **syncrepl** directive within the replication consumer site. It is a non-negative integer having no more than three digits.

provider specifies the replication provider site containing the master content as an LDAP URI. If <port> is not given, the standard LDAP port number (389 or 636) is used.

The content of the **syncrepl** replica is defined using a search specification as its result set. The consumer **slapd** will send search requests to the provider **slapd** according to the search specification. The search specification includes **searchbase, scope, filter, attrs, attrsonly, sizelimit,** and **timelimit** parameters as in the normal search specification. The **exattrs** option may also be used to specify attributes that should be omitted from incoming entries. The **scope** defaults to **sub**, the **filter** defaults to **(objectclass=*)**, and there is no default **searchbase**. The **attrs** list defaults to **"*,+"** to return all user and operational attributes, and **attrsonly** and **exattrs** are unset by default. The **sizelimit** and **timelimit** only accept "unlimited" and positive integers, and both default to "unlimited". Note, however, that any provider-side limits for the replication identity will be enforced by the provider regardless of the limits requested by the LDAP Content Synchronization operation, much like for any other search operation.

The LDAP Content Synchronization protocol has two operation types. In the **refreshOnly** operation, the next synchronization search operation is periodically rescheduled at an interval time (specified by **interval** parameter; 1 day by default) after each synchronization operation finishes. In the **refreshAndPersist** operation, a synchronization search remains persistent in the provider slapd. Further updates to the master replica will generate **searchResultEntry** to the consumer slapd as the search responses to the persistent synchronization search.

If an error occurs during replication, the consumer will attempt to reconnect according to the **retry** parameter which is a list of the <retry interval> and <# of retries> pairs. For example, retry="60 10 300 3" lets the consumer retry every 60 seconds for the first 10 times and then retry every 300 seconds for the next 3 times before stop retrying. The ‘+’ in <# of retries> means indefinite number of retries until success.

The schema checking can be enforced at the LDAP Sync consumer site by turning on the **schemachecking** parameter. The default is off.

A **bindmethod** of **simple** requires the options **binddn** and **credentials** and should only be used when adequate security services (e.g. TLS or IPSEC) are in place. A **bindmethod** of **sasl** requires the option **saslmech.** Depending on the mechanism, an authentication identity and/or credentials can be specified using **authcid** and **credentials.** The **authzid** parameter may be used to specify an authorization identity. Specific security properties (as with the **sasl-secprops** keyword above) for a SASL bind can be set with the **secprops** option. A non default SASL realm can be set with the **realm** option. The provider, other than allow authentication of the syncrepl identity, should grant that identity appropriate access privileges to the data that is being replicated (**access** directive), and appropriate time and size limits (**limits** directive).

The **starttls** parameter specifies use of the StartTLS extended operation to establish a TLS session before Binding to the provider. If the **critical** argument is supplied, the session will be aborted if the StartTLS request fails. Otherwise the syncrepl session continues without TLS. The tls_reqcert setting defaults to "demand" and the other TLS settings default to the same as the main slapd TLS settings.

Rather than replicating whole entries, the consumer can query logs of data modifications. This

mode of operation is referred to as *delta syncrepl*. In addition to the above parameters, the **logbase** and **logfilter** parameters must be set appropriately for the log that will be used. The **syncdata** parameter must be set to either "accesslog" if the log conforms to the **slapo-accesslog**(5) log format, or "changelog" if the log conforms to the obsolete *changelog* format. If the **syncdata** parameter is omitted or set to "default" then the log parameters are ignored.

olcUpdateDN: <dn>

This option is only applicable in a slave database. It specifies the DN permitted to update (subject to access controls) the replica. It is only needed in certain push-mode replication scenarios. Generally, this DN *should not* be the same as the **rootdn** used at the master.

olcUpdateRef: <url>

Specify the referral to pass back when **slapd**(8) is asked to modify a replicated local database. If multiple values are specified, each url is provided.

DATABASE-SPECIFIC OPTIONS

Each database may allow specific configuration options; they are documented separately in the backends' manual pages. See the **slapd.backends**(5) manual page for an overview of available backends.

OVERLAYS

An overlay is a piece of code that intercepts database operations in order to extend or change them. Overlays are pushed onto a stack over the database, and so they will execute in the reverse of the order in which they were configured and the database itself will receive control last of all.

Overlays must be configured as child entries of a specific database. The entry's RDN must be of the form **olcOverlay={x}<overlaytype>** and the entry must have the olcOverlayConfig objectClass. Normally the config engine generates the "{x}" index in the RDN automatically, so it can be omitted when initially loading these entries.

See the **slapd.overlays**(5) manual page for an overview of available overlays.

EXAMPLES

Here is a short example of a configuration in LDIF suitable for use with **slapadd**(8) :

```
dn: cn=config
objectClass: olcGlobal
cn: config
olcPidFile: /var/openldap/run/slapd.pid
olcAttributeOptions: x-hidden lang-

dn: cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema

include: /etc/openldap/schema/core.ldif

dn: olcDatabase=frontend,cn=config
objectClass: olcDatabaseConfig
objectClass: olcFrontendConfig
olcDatabase: frontend
# Subtypes of "name" (e.g. "cn" and "ou") with the
# option ";x-hidden" can be searched for/compared,
# but are not shown.  See slapd.access(5).
olcAccess: to attrs=name;x-hidden by * =cs
# Protect passwords.  See slapd.access(5).
olcAccess: to attrs=userPassword  by * auth
# Read access to other attributes and entries.
```

```
olcAccess: to * by * read

# set a rootpw for the config database so we can bind.
# deny access to everyone else.
dn: olcDatabase=config,cn=config
objectClass: olcDatabaseConfig
olcDatabase: config
olcRootPW: {SSHA}XKYnrjvGT3wZFQrDD5040US592LxsdLy
olcAccess: to * by * none

dn: olcDatabase=bdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcBdbConfig
olcDatabase: bdb
olcSuffix: "dc=our-domain,dc=com"
# The database directory MUST exist prior to
# running slapd AND should only be accessible
# by the slapd/tools. Mode 0700 recommended.
olcDbDirectory: /var/openldap/openldap-data
# Indices to maintain
olcDbIndex:     objectClass  eq
olcDbIndex:     cn,sn,mail   pres,eq,approx,sub

# We serve small clients that do not handle referrals,
# so handle remote lookups on their behalf.
dn: olcDatabase=ldap,cn=config
objectClass: olcDatabaseConfig
objectClass: olcLdapConfig
olcDatabase: ldap
olcSuffix: ""
olcDbUri: ldap://ldap.some-server.com/
```

Assuming the above data was saved in a file named "config.ldif" and the /etc/openldap/slapd.d directory has been created, this command will initialize the configuration:

```
slapadd -F /etc/openldap/slapd.d -n 0 -l config.ldif
```

"OpenLDAP Administrator’s Guide" contains a longer annotated example of a slapd configuration.

Alternatively, an existing slapd.conf file can be converted to the new format using slapd or any of the slap tools:

```
slaptest -f /etc/openldap/slapd.conf -F /etc/openldap/slapd.d
```

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

/etc/openldap/slapd.d
: default slapd configuration directory

SEE ALSO

ldap(3), **ldif**(5), **slapd.access**(5), **slapd.backends**(5), **slapd.conf**(5), **slapd.overlays**(5), **slapd.plugin**(5), **slapd.replog**(5), **slapd**(8), **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), **slappasswd**(8), **slaptest**(8).

"OpenLDAP Administrator’s Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapd-dnssrv − DNS SRV referral backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The DNSSRV backend to **slapd**(8) serves up referrals based upon SRV resource records held in the Domain Name System.

This backend is experimental.

CONFIGURATION

The DNSSRV backend has no backend nor database specific options. It is configured simply by "database dnssrv" followed a suffix directive, e.g. suffix "".

ACCESS CONTROL

The **dnssrv** backend does not honor all ACL semantics as described in **slapd.access**(5). In fact, this backend only implements the **search** operation when the **manageDSAit** control (RFC 3296) is used, otherwise for every operation a referral, whenever appropriate, or an error is returned. Currently, there is no means to condition the returning of the referral by means of ACLs; no access control is implemented, except for **read (=r)** access to the returned entries, which is actually provided by the frontend. Note, however, that the information returned by this backend is collected through the DNS, so it is public by definition.

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

"OpenLDAP Root Service - An experimental LDAP referral service" [RFC 3088],
"OpenLDAP LDAP Root Service" <http://www.openldap.org/faq/?file=393)>,
slapd.conf(5), **slapd**(8)

NAME

slapd-dnssrv − DNS SRV referral backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The DNSSRV backend to **slapd**(8) serves up referrals based upon SRV resource records held in the Domain Name System.

This backend is experimental.

CONFIGURATION

The DNSSRV backend has no backend nor database specific options. It is configured simply by "database dnssrv" followed a suffix directive, e.g. suffix "".

ACCESS CONTROL

The **dnssrv** backend does not honor all ACL semantics as described in **slapd.access**(5). In fact, this backend only implements the **search** operation when the **manageDSAit** control (RFC 3296) is used, otherwise for every operation a referral, whenever appropriate, or an error is returned. Currently, there is no means to condition the returning of the referral by means of ACLs; no access control is implemented, except for **read (=r)** access to the returned entries, which is actually provided by the frontend. Note, however, that the information returned by this backend is collected through the DNS, so it is public by definition.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

"OpenLDAP Root Service - An experimental LDAP referral service" [RFC 3088],
"OpenLDAP LDAP Root Service" <http://www.openldap.org/faq/?file=393)>,
slapd.conf(5), **slapd**(8)

NAME

slapd-ldap − LDAP backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The LDAP backend to **slapd**(8) is not an actual database; instead it acts as a proxy to forward incoming requests to another LDAP server. While processing requests it will also chase referrals, so that referrals are fully processed instead of being returned to the slapd client.

Sessions that explicitly Bind to the back-ldap database always create their own private connection to the remote LDAP server. Anonymous sessions will share a single anonymous connection to the remote server. For sessions bound through other mechanisms, all sessions with the same DN will share the same connection. This connection pooling strategy can enhance the proxy's efficiency by reducing the overhead of repeatedly making/breaking multiple connections.

The ldap database can also act as an information service, i.e. the identity of locally authenticated clients is asserted to the remote server, possibly in some modified form. For this purpose, the proxy binds to the remote server with some administrative identity, and, if required, authorizes the asserted identity. See the *idassert-** rules below. The administrative identity of the proxy, on the remote server, must be allowed to authorize by means of appropriate **authzTo** rules; see **slapd.conf**(5) for details.

Note: When looping back to the same instance of **slapd**(8), each connection requires a new thread; as a consequence, **slapd**(8) must be compiled with thread support, and the **threads** parameter may need some tuning; in those cases, one may consider using **slapd-relay**(5) instead, which performs the relayed operation internally and thus reuses the same connection.

CONFIGURATION

These **slapd.conf** options apply to the LDAP backend database. That is, they must follow a "database ldap" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

Note: In early versions of back-ldap it was recommended to always set

lastmod off

for **ldap** and **meta** databases. This was required because operational attributes related to entry creation and modification should not be proxied, as they could be mistakenly written to the target server(s), generating an error. The current implementation automatically sets lastmod to **off**, so its use is redundant and should be omitted.

uri <ldapurl>

LDAP server to use. Multiple URIs can be set in a single **ldapurl** argument, resulting in the underlying library automatically call the first server of the list that responds, e.g.

uri "ldap://host/ ldap://backup-host/"

The URI list is space- or comma-separated. Whenever the server that responds is not the first one in the list, the list is rearranged and the responsive server is moved to the head, so that it will be first contacted the next time a connection needs be created.

acl-bind bindmethod=simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>]

[tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Allows to define the parameters of the authentication method that is internally used by the proxy to collect info related to access control, and whenever an operation occurs with the identity of the rootdn of the LDAP proxy database. The identity defined by this directive, according to the properties associated to the authentication method, is supposed to have read access on the target server to attributes used on the proxy for ACL checking.

There is no risk of giving away such values; they are only used to check permissions. The default is to use **simple** bind, with empty *binddn* and *credentials*, which means that the related operations will be performed anonymously. If not set, and if **idassert-bind** is defined, this latter identity is used instead. See **idassert-bind** for details.

The connection between the proxy database and the remote server associated to this identity is cached regardless of the lifespan of the client-proxy connection that first established it.

This identity is by no means implicitly used by the proxy when the client connects anonymously. The **idassert-bind** feature, instead, in some cases can be crafted to implement that behavior, which is *intrinsically unsafe and should be used with extreme care*. This directive obsoletes **acl-authcDN**, and **acl-passwd**.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

cancel {ABANDON|ignore|exop[-discover]}
Defines how to handle operation cancellation. By default, **abandon** is invoked, so the operation is abandoned immediately. If set to **ignore**, no action is taken and any further response is ignored; this may result in further response messages to be queued for that connection, so it is recommended that long lasting connections are timed out either by *idle-timeout* or *conn-ttl*, so that resources eventually get released. If set to **exop**, a *cancel* operation (RFC 3909) is issued, resulting in the cancellation of the current operation; the *cancel* operation waits for remote server response, so its use may not be recommended. If set to **exop-discover**, support of the *cancel* extended operation is detected by reading the remote server's root DSE.

chase-referrals {YES|no}
enable/disable automatic referral chasing, which is delegated to the underlying libldap, with rebinding eventually performed if the **rebind-as-user** directive is used. The default is to chase referrals.

conn-ttl <time>
This directive causes a cached connection to be dropped an recreated after a given ttl, regardless of being idle or not.

idassert-authzFrom <authz-regexp>
if defined, selects what *local* identities are authorized to exploit the identity assertion feature. The string **<authz-regexp>** follows the rules defined for the *authzFrom* attribute. See **slapd.conf**(5), section related to **authz-policy**, for details on the syntax of this field.

idassert-bind bindmethod=none|simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [authz={native|proxyauthz}] [mode=<mode>] [flags=<flags>] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>]

[tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Allows to define the parameters of the authentication method that is internally used by the proxy to authorize connections that are authenticated by other databases. The identity defined by this directive, according to the properties associated to the authentication method, is supposed to have auth access on the target server to attributes used on the proxy for authentication and authorization, and to be allowed to authorize the users. This requires to have **proxyAuthz** privileges on a wide set of DNs, e.g. **authzTo=dn.subtree:""**, and the remote server to have **authz-policy** set to **to** or **both**. See **slapd.conf**(5) for details on these statements and for remarks and drawbacks about their usage. The supported bindmethods are

none|simple|sasl

where **none** is the default, i.e. no *identity assertion* is performed.

The authz parameter is used to instruct the SASL bind to exploit **native** SASL authorization, if available; since connections are cached, this should only be used when authorizing with a fixed identity (e.g. by means of the **authzDN** or **authzID** parameters). Otherwise, the default **proxyauthz** is used, i.e. the proxyAuthz control (Proxied Authorization, RFC 4370) is added to all operations.

The supported modes are:

<mode> := {legacy|anonymous|none|self}

If **<mode>** is not present, and **authzId** is given, the proxy always authorizes that identity. **<authorization ID>** can be

u:<user>

[dn:]<DN>

The former is supposed to be expanded by the remote server according to the authz rules; see **slapd.conf**(5) for details. In the latter case, whether or not the **dn:** prefix is present, the string must pass DN validation and normalization.

The default mode is **legacy**, which implies that the proxy will either perform a simple bind as the *authcDN* or a SASL bind as the *authcID* and assert the client's identity when it is not anonymous. Direct binds are always proxied. The other modes imply that the proxy will always either perform a simple bind as the *authcDN* or a SASL bind as the *authcID*, unless restricted by **idassert-authzFrom** rules (see below), in which case the operation will fail; eventually, it will assert some other identity according to **<mode>**. Other identity assertion modes are **anonymous** and **self**, which respectively mean that the *empty* or the *client*'s identity will be asserted; **none**, which means that no proxyAuthz control will be used, so the *authcDN* or the *authcID* identity will be asserted. For all modes that require the use of the *proxyAuthz* control, on the remote server the proxy identity must have appropriate *authzTo* permissions, or the asserted identities must have appropriate *authzFrom* permissions. Note, however, that the ID assertion feature is mostly useful when the asserted identities do not exist on the remote server.

Flags can be

override,[non-]prescriptive

When the **override** flag is used, identity assertion takes place even when the database is

authorizing for the identity of the client, i.e. after binding with the provided identity, and thus authenticating it, the proxy performs the identity assertion using the configured identity and authentication method.

When the **prescriptive** flag is used (the default), operations fail with *inappropriateAuthentication* for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns. If the **non-prescriptive** flag is used, operations are performed anonymously for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

The identity associated to this directive is also used for privileged operations whenever **idassert-bind** is defined and **acl-bind** is not. See **acl-bind** for details.

This directive obsoletes **idassert-authcDN**, **idassert-passwd**, **idassert-mode**, and **idassert-method**.

idle-timeout <time>

This directive causes a cached connection to be dropped an recreated after it has been idle for the specified time.

network-timeout <time>

Sets the network timeout value after which **poll**(2)/**select**(2) following a **connect**(2) returns in case of no activity. The value is in seconds, and it can be specified as for **idle-timeout**.

protocol−version {0,2,3}

This directive indicates what protocol version must be used to contact the remote server. If set to 0 (the default), the proxy uses the same protocol version used by the client, otherwise the requested protocol is used. The proxy returns *unwillingToPerform* if an operation that is incompatible with the requested protocol is attempted.

proxy−whoami {NO|yes}

Turns on proxying of the WhoAmI extended operation. If this option is given, back-ldap will replace slapd's original WhoAmI routine with its own. On slapd sessions that were authenticated by back-ldap, the WhoAmI request will be forwarded to the remote LDAP server. Other sessions will be handled by the local slapd, as before. This option is mainly useful in conjunction with Proxy Authorization.

quarantine <interval>,<num>[;<interval>,<num>[...]]

Turns on quarantine of URIs that returned *LDAP_UNAVAILABLE*, so that an attempt to reconnect only occurs at given intervals instead of any time a client requests an operation. The pattern is: retry only after at least *interval* seconds elapsed since last attempt, for exactly *num* times; then use the next pattern. If *num* for the last pattern is "+", it retries forever; otherwise, no more retries occur. The process can be restarted by resetting the *olcDbQuarantine* attribute of the database entry in the configuration backend.

rebind-as-user {NO|yes}

If this option is given, the client's bind credentials are remembered for rebinds, when trying to re-establish a broken connection, or when chasing a referral, if **chase-referrals** is set to *yes*.

session−tracking−request {NO|yes}
Adds session tracking control for all requests. The client's IP and hostname, and the identity associated to each request, if known, are sent to the remote server for informational purposes. This directive is incompatible with setting *protocol−version* to 2.

single−conn {NO|yes}
Discards current cached connection when the client rebinds.

t-f-support {NO|yes|discover}
enable if the remote server supports absolute filters (see *draft-zeilenga-ldap-t-f* for details). If set to **discover**, support is detected by reading the remote server's root DSE.

timeout [<op>=]<val> [...]
This directive allows to set per-operation timeouts. Operations can be

<op> ::= bind, add, delete, modrdn, modify, compare, search

The overall duration of the **search** operation is controlled either by the **timelimit** parameter or by server-side enforced time limits (see **timelimit** and **limits** in **slapd.conf**(5) for details). This **timeout** parameter controls how long the target can be irresponsive before the operation is aborted. Timeout is meaningless for the remaining operations, **unbind** and **abandon**, which do not imply any response, while it is not yet implemented in currently supported **extended** operations. If no operation is specified, the timeout **val** affects all supported operations.

Note: if the timelimit is exceeded, the operation is cancelled (according to the **cancel** directive); the protocol does not provide any means to rollback operations, so the client will not be notified about the result of the operation, which may eventually succeeded or not. In case the timeout is exceeded during a bind operation, the connection is destroyed, according to RFC4511.

Note: in some cases, this backend may issue binds prior to other operations (e.g. to bind anonymously or with some prescribed identity according to the **idassert-bind** directive). In this case, the timeout of the operation that resulted in the bind is used.

tls {[try-]start|[try-]propagate|ldaps} [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Specify the use of TLS when a regular connection is initialized. The StartTLS extended operation will be used unless the URI directive protocol scheme is **ldaps://**. In that case this keyword may only be set to "ldaps" and the StartTLS operation will not be used. **propagate** issues the StartTLS operation only if the original connection did. The **try-** prefix instructs the proxy to continue operations if the StartTLS operation failed; its use is **not** recommended.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

use-temporary-conn {NO|yes}
when set to **yes**, create a temporary connection whenever competing with other threads for a shared one; otherwise, wait until the shared connection is available.

BACKWARD COMPATIBILITY

The LDAP backend has been heavily reworked between releases 2.2 and 2.3, and subsequently between 2.3 and 2.4. As a side-effect, some of the traditional directives have been deprecated and should be no longer

used, as they might disappear in future releases.

acl-authcDN <administrative DN for access control purposes>
Formerly known as the **binddn**, it is the DN that is used to query the target server for acl checking; it is supposed to have read access on the target server to attributes used on the proxy for acl checking. There is no risk of giving away such values; they are only used to check permissions.

The acl-authcDN identity is by no means implicitly used by the proxy when the client connects anonymously. The **idassert-*** feature can be used (at own risk) for that purpose instead.

This directive is obsoleted by the **binddn** arg of **acl-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

acl-passwd <password>
Formerly known as the **bindpw**, it is the password used with the above **acl-authcDN** directive. This directive is obsoleted by the **credentials** arg of **acl-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

idassert-authcDN <administrative DN for proxyAuthz purposes>
DN which is used to propagate the client's identity to the target by means of the proxyAuthz control when the client does not belong to the DIT fragment that is being proxied by back-ldap. This directive is obsoleted by the **binddn** arg of **idassert-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

idassert-passwd <password>
Password used with the **idassert-authcDN** above. This directive is obsoleted by the **crendentials** arg of **idassert-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

idassert-mode <mode> [<flags>]
defines what type of *identity assertion* is used. This directive is obsoleted by the **mode** arg of **idassert-bind**, and will be dismissed in the future.

idassert-method <method> [<saslargs>]
This directive is obsoleted by the **bindmethod** arg of **idassert-bind**, and will be dismissed in the future.

port <port>
this directive is no longer supported. Use the **uri** directive as described above.

server <hostname[:port]>
this directive is no longer supported. Use the **uri** directive as described above.

suffixmassage, map, rewrite*
These directives are no longer supported by back-ldap; their functionality is now delegated to the **rwm** overlay. Essentially, add a statement

overlay rwm

first, and prefix all rewrite/map statements with **rwm-** to obtain the original behavior. See **slapo-rwm**(5) for details.

ACCESS CONTROL

The **ldap** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access checking is delegated to the remote server(s). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

OVERLAYS

The LDAP backend provides basic proxying functionalities to many overlays. The **chain** overlay, described in **slapo−chain**(5), and the **translucent** overlay, described in **slapo−translucent**(5), deserve a special mention.

Conversely, there are many overlays that are best used in conjunction with the LDAP backend. The **proxycache** overlay allows caching of LDAP search requests (queries) in a local database. See **slapo−pcache**(5) for details. The **rwm** overlay provides DN rewrite and attribute/objectClass mapping capabilities to the underlying database. See **slapo−rwm**(5) for details.

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−meta**(5), **slapo−chain**(5), **slapo−pcache**(5), **slapo−rwm**(5), **slapo−translucent**(5), **slapd**(8), **ldap**(3).

AUTHOR

Howard Chu, with enhancements by Pierangelo Masarati

NAME

slapd-ldap − LDAP backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The LDAP backend to **slapd**(8) is not an actual database; instead it acts as a proxy to forward incoming requests to another LDAP server. While processing requests it will also chase referrals, so that referrals are fully processed instead of being returned to the slapd client.

Sessions that explicitly Bind to the back-ldap database always create their own private connection to the remote LDAP server. Anonymous sessions will share a single anonymous connection to the remote server. For sessions bound through other mechanisms, all sessions with the same DN will share the same connection. This connection pooling strategy can enhance the proxy's efficiency by reducing the overhead of repeatedly making/breaking multiple connections.

The ldap database can also act as an information service, i.e. the identity of locally authenticated clients is asserted to the remote server, possibly in some modified form. For this purpose, the proxy binds to the remote server with some administrative identity, and, if required, authorizes the asserted identity. See the *idassert-** rules below. The administrative identity of the proxy, on the remote server, must be allowed to authorize by means of appropriate **authzTo** rules; see **slapd.conf**(5) for details.

Note: When looping back to the same instance of **slapd**(8), each connection requires a new thread; as a consequence, **slapd**(8) must be compiled with thread support, and the **threads** parameter may need some tuning; in those cases, one may consider using **slapd-relay**(5) instead, which performs the relayed operation internally and thus reuses the same connection.

CONFIGURATION

These **slapd.conf** options apply to the LDAP backend database. That is, they must follow a "database ldap" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

Note: In early versions of back-ldap it was recommended to always set

lastmod off

for **ldap** and **meta** databases. This was required because operational attributes related to entry creation and modification should not be proxied, as they could be mistakenly written to the target server(s), generating an error. The current implementation automatically sets lastmod to **off**, so its use is redundant and should be omitted.

uri <ldapurl>

LDAP server to use. Multiple URIs can be set in a single **ldapurl** argument, resulting in the underlying library automatically call the first server of the list that responds, e.g.

uri "ldap://host/ ldap://backup-host/"

The URI list is space- or comma-separated. Whenever the server that responds is not the first one in the list, the list is rearranged and the responsive server is moved to the head, so that it will be first contacted the next time a connection needs be created.

acl-bind bindmethod=simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>]

[tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Allows to define the parameters of the authentication method that is internally used by the proxy to collect info related to access control, and whenever an operation occurs with the identity of the rootdn of the LDAP proxy database. The identity defined by this directive, according to the properties associated to the authentication method, is supposed to have read access on the target server to attributes used on the proxy for ACL checking.

There is no risk of giving away such values; they are only used to check permissions. The default is to use **simple** bind, with empty *binddn* and *credentials*, which means that the related operations will be performed anonymously. If not set, and if **idassert-bind** is defined, this latter identity is used instead. See **idassert-bind** for details.

The connection between the proxy database and the remote server associated to this identity is cached regardless of the lifespan of the client-proxy connection that first established it.

This identity is by no means implicitly used by the proxy when the client connects anonymously. The **idassert-bind** feature, instead, in some cases can be crafted to implement that behavior, which is *intrinsically unsafe and should be used with extreme care*. This directive obsoletes **acl-authcDN**, and **acl-passwd**.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

cancel {ABANDON|ignore|exop[-discover]}
Defines how to handle operation cancellation. By default, **abandon** is invoked, so the operation is abandoned immediately. If set to **ignore**, no action is taken and any further response is ignored; this may result in further response messages to be queued for that connection, so it is recommended that long lasting connections are timed out either by *idle-timeout* or *conn-ttl*, so that resources eventually get released. If set to **exop**, a *cancel* operation (RFC 3909) is issued, resulting in the cancellation of the current operation; the *cancel* operation waits for remote server response, so its use may not be recommended. If set to **exop-discover**, support of the *cancel* extended operation is detected by reading the remote server's root DSE.

chase-referrals {YES|no}
enable/disable automatic referral chasing, which is delegated to the underlying libldap, with rebinding eventually performed if the **rebind-as-user** directive is used. The default is to chase referrals.

conn-ttl <time>
This directive causes a cached connection to be dropped an recreated after a given ttl, regardless of being idle or not.

idassert-authzFrom <authz-regexp>
if defined, selects what *local* identities are authorized to exploit the identity assertion feature. The string **<authz-regexp>** follows the rules defined for the *authzFrom* attribute. See **slapd.conf**(5), section related to **authz-policy**, for details on the syntax of this field.

idassert-bind bindmethod=none|simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [authz={native|proxyauthz}] [mode=<mode>] [flags=<flags>] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>]

[tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Allows to define the parameters of the authentication method that is internally used by the proxy to authorize connections that are authenticated by other databases. The identity defined by this directive, according to the properties associated to the authentication method, is supposed to have auth access on the target server to attributes used on the proxy for authentication and authorization, and to be allowed to authorize the users. This requires to have **proxyAuthz** privileges on a wide set of DNs, e.g. **authzTo=dn.subtree:""**, and the remote server to have **authz-policy** set to **to** or **both**. See **slapd.conf**(5) for details on these statements and for remarks and drawbacks about their usage. The supported bindmethods are

none|simple|sasl

where **none** is the default, i.e. no *identity assertion* is performed.

The authz parameter is used to instruct the SASL bind to exploit **native** SASL authorization, if available; since connections are cached, this should only be used when authorizing with a fixed identity (e.g. by means of the **authzDN** or **authzID** parameters). Otherwise, the default **proxyauthz** is used, i.e. the proxyAuthz control (Proxied Authorization, RFC 4370) is added to all operations.

The supported modes are:

<mode> := {legacy|anonymous|none|self}

If **<mode>** is not present, and **authzId** is given, the proxy always authorizes that identity. **<authorization ID>** can be

u:<user>

[dn:]<DN>

The former is supposed to be expanded by the remote server according to the authz rules; see **slapd.conf**(5) for details. In the latter case, whether or not the **dn:** prefix is present, the string must pass DN validation and normalization.

The default mode is **legacy**, which implies that the proxy will either perform a simple bind as the *authcDN* or a SASL bind as the *authcID* and assert the client's identity when it is not anonymous. Direct binds are always proxied. The other modes imply that the proxy will always either perform a simple bind as the *authcDN* or a SASL bind as the *authcID*, unless restricted by **idassert-authzFrom** rules (see below), in which case the operation will fail; eventually, it will assert some other identity according to **<mode>**. Other identity assertion modes are **anonymous** and **self**, which respectively mean that the *empty* or the *client*'s identity will be asserted; **none**, which means that no proxyAuthz control will be used, so the *authcDN* or the *authcID* identity will be asserted. For all modes that require the use of the *proxyAuthz* control, on the remote server the proxy identity must have appropriate *authzTo* permissions, or the asserted identities must have appropriate *authzFrom* permissions. Note, however, that the ID assertion feature is mostly useful when the asserted identities do not exist on the remote server.

Flags can be

override,[non-]prescriptive

When the **override** flag is used, identity assertion takes place even when the database is

authorizing for the identity of the client, i.e. after binding with the provided identity, and thus authenticating it, the proxy performs the identity assertion using the configured identity and authentication method.

When the **prescriptive** flag is used (the default), operations fail with *inappropriateAuthentication* for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns. If the **non-prescriptive** flag is used, operations are performed anonymously for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

The identity associated to this directive is also used for privileged operations whenever **idassert-bind** is defined and **acl-bind** is not. See **acl-bind** for details.

This directive obsoletes **idassert-authcDN**, **idassert-passwd**, **idassert-mode**, and **idassert-method**.

idle-timeout <time>

This directive causes a cached connection to be dropped an recreated after it has been idle for the specified time.

network-timeout <time>

Sets the network timeout value after which **poll**(2)/**select**(2) following a **connect**(2) returns in case of no activity. The value is in seconds, and it can be specified as for **idle-timeout**.

protocol−version {0,2,3}

This directive indicates what protocol version must be used to contact the remote server. If set to 0 (the default), the proxy uses the same protocol version used by the client, otherwise the requested protocol is used. The proxy returns *unwillingToPerform* if an operation that is incompatible with the requested protocol is attempted.

proxy−whoami {NO|yes}

Turns on proxying of the WhoAmI extended operation. If this option is given, back-ldap will replace slapd's original WhoAmI routine with its own. On slapd sessions that were authenticated by back-ldap, the WhoAmI request will be forwarded to the remote LDAP server. Other sessions will be handled by the local slapd, as before. This option is mainly useful in conjunction with Proxy Authorization.

quarantine <interval>,<num>[;<interval>,<num>[...]]

Turns on quarantine of URIs that returned *LDAP_UNAVAILABLE*, so that an attempt to reconnect only occurs at given intervals instead of any time a client requests an operation. The pattern is: retry only after at least *interval* seconds elapsed since last attempt, for exactly *num* times; then use the next pattern. If *num* for the last pattern is "+", it retries forever; otherwise, no more retries occur. The process can be restarted by resetting the *olcDbQuarantine* attribute of the database entry in the configuration backend.

rebind-as-user {NO|yes}

If this option is given, the client's bind credentials are remembered for rebinds, when trying to re-establish a broken connection, or when chasing a referral, if **chase-referrals** is set to *yes*.

session−tracking−request {NO|yes}
Adds session tracking control for all requests. The client's IP and hostname, and the identity associated to each request, if known, are sent to the remote server for informational purposes. This directive is incompatible with setting *protocol−version* to 2.

single−conn {NO|yes}
Discards current cached connection when the client rebinds.

t-f-support {NO|yes|discover}
enable if the remote server supports absolute filters (see *draft-zeilenga-ldap-t-f* for details). If set to **discover**, support is detected by reading the remote server's root DSE.

timeout [<op>=]<val> [...]
This directive allows to set per-operation timeouts. Operations can be

<op> ::= bind, add, delete, modrdn, modify, compare, search

The overall duration of the **search** operation is controlled either by the **timelimit** parameter or by server-side enforced time limits (see **timelimit** and **limits** in **slapd.conf**(5) for details). This **timeout** parameter controls how long the target can be irresponsive before the operation is aborted. Timeout is meaningless for the remaining operations, **unbind** and **abandon**, which do not imply any response, while it is not yet implemented in currently supported **extended** operations. If no operation is specified, the timeout **val** affects all supported operations.

Note: if the timelimit is exceeded, the operation is cancelled (according to the **cancel** directive); the protocol does not provide any means to rollback operations, so the client will not be notified about the result of the operation, which may eventually succeeded or not. In case the timeout is exceeded during a bind operation, the connection is destroyed, according to RFC4511.

Note: in some cases, this backend may issue binds prior to other operations (e.g. to bind anonymously or with some prescribed identity according to the **idassert-bind** directive). In this case, the timeout of the operation that resulted in the bind is used.

tls {[try-]start|[try-]propagate|ldaps} [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Specify the use of TLS when a regular connection is initialized. The StartTLS extended operation will be used unless the URI directive protocol scheme is **ldaps://**. In that case this keyword may only be set to "ldaps" and the StartTLS operation will not be used. **propagate** issues the StartTLS operation only if the original connection did. The **try-** prefix instructs the proxy to continue operations if the StartTLS operation failed; its use is **not** recommended.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

use-temporary-conn {NO|yes}
when set to **yes**, create a temporary connection whenever competing with other threads for a shared one; otherwise, wait until the shared connection is available.

BACKWARD COMPATIBILITY

The LDAP backend has been heavily reworked between releases 2.2 and 2.3, and subsequently between 2.3 and 2.4. As a side-effect, some of the traditional directives have been deprecated and should be no longer

used, as they might disappear in future releases.

acl-authcDN <administrative DN for access control purposes>
Formerly known as the **binddn**, it is the DN that is used to query the target server for acl checking; it is supposed to have read access on the target server to attributes used on the proxy for acl checking. There is no risk of giving away such values; they are only used to check permissions.

The acl-authcDN identity is by no means implicitly used by the proxy when the client connects anonymously. The **idassert-*** feature can be used (at own risk) for that purpose instead.

This directive is obsoleted by the **binddn** arg of **acl-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

acl-passwd <password>
Formerly known as the **bindpw**, it is the password used with the above **acl-authcDN** directive. This directive is obsoleted by the **credentials** arg of **acl-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

idassert-authcDN <administrative DN for proxyAuthz purposes>
DN which is used to propagate the client's identity to the target by means of the proxyAuthz control when the client does not belong to the DIT fragment that is being proxied by back-ldap. This directive is obsoleted by the **binddn** arg of **idassert-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

idassert-passwd <password>
Password used with the **idassert-authcDN** above. This directive is obsoleted by the **crendentials** arg of **idassert-bind** when *bindmethod*=**simple**, and will be dismissed in the future.

idassert-mode <mode> [<flags>]
defines what type of *identity assertion* is used. This directive is obsoleted by the **mode** arg of **idassert-bind**, and will be dismissed in the future.

idassert-method <method> [<saslargs>]
This directive is obsoleted by the **bindmethod** arg of **idassert-bind**, and will be dismissed in the future.

port <port>
this directive is no longer supported. Use the **uri** directive as described above.

server <hostname[:port]>
this directive is no longer supported. Use the **uri** directive as described above.

suffixmassage, map, rewrite*
These directives are no longer supported by back-ldap; their functionality is now delegated to the **rwm** overlay. Essentially, add a statement

overlay rwm

first, and prefix all rewrite/map statements with **rwm-** to obtain the original behavior. See **slapo-rwm**(5) for details.

ACCESS CONTROL

The **ldap** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access checking is delegated to the remote server(s). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

OVERLAYS

The LDAP backend provides basic proxying functionalities to many overlays. The **chain** overlay, described in **slapo−chain**(5), and the **translucent** overlay, described in **slapo−translucent**(5), deserve a special mention.

Conversely, there are many overlays that are best used in conjunction with the LDAP backend. The **proxycache** overlay allows caching of LDAP search requests (queries) in a local database. See **slapo−pcache**(5) for details. The **rwm** overlay provides DN rewrite and attribute/objectClass mapping capabilities to the underlying database. See **slapo−rwm**(5) for details.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−meta**(5), **slapo−chain**(5), **slapo−pcache**(5), **slapo−rwm**(5), **slapo−translucent**(5), **slapd**(8), **ldap**(3).

AUTHOR

Howard Chu, with enhancements by Pierangelo Masarati

NAME

slapd-ldbm − Discontinued LDBM backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

LDBM was the original database backend to **slapd**(8), and was supported up to OpenLDAP 2.3. It has been superseded by the more robust BDB and HDB backends.

SEE ALSO

slapd(8), **slapd-bdb**(5), **slapd.backends**(5).

ACKNOWLEDGEMENTS

NAME

slapd-ldbm − Discontinued LDBM backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

LDBM was the original database backend to **slapd**(8), and was supported up to OpenLDAP 2.3. It has been superseded by the more robust BDB and HDB backends.

SEE ALSO

slapd(8), **slapd-bdb**(5), **slapd.backends**(5).

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapd-ldif − LDIF backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The LDIF backend to **slapd**(8) is a basic storage backend that stores entries in text files in LDIF format, and exploits the filesystem to create the tree structure of the database. It is intended as a cheap, low performance easy to use backend, and it is exploited by higher-level internal structures to provide a permanent storage.

CONFIGURATION

These **slapd.conf** options apply to the LDIF backend database. That is, they must follow a "database ldif" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

directory <dir>

Specify the directory where the database tree starts. The directory must exist and grant appropriate permissions (rwx) to the identity slapd is running with.

ACCESS CONTROL

The **LDIF** backend does not honor any of the access control semantics described in **slapd.access**(5). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **ldif**(5).

AUTHOR

Eric Stokes

NAME

slapd-ldif − LDIF backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The LDIF backend to **slapd**(8) is a basic storage backend that stores entries in text files in LDIF format, and exploits the filesystem to create the tree structure of the database. It is intended as a cheap, low performance easy to use backend, and it is exploited by higher-level internal structures to provide a permanent storage.

CONFIGURATION

These **slapd.conf** options apply to the LDIF backend database. That is, they must follow a "database ldif" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

directory <dir>

Specify the directory where the database tree starts. The directory must exist and grant appropriate permissions (rwx) to the identity slapd is running with.

ACCESS CONTROL

The **LDIF** backend does not honor any of the access control semantics described in **slapd.access**(5). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

FILES

/etc/openldap/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **ldif**(5).

AUTHOR

Eric Stokes

NAME

slapd-meta − metadirectory backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **meta** backend to **slapd**(8) performs basic LDAP proxying with respect to a set of remote LDAP servers, called "targets". The information contained in these servers can be presented as belonging to a single Directory Information Tree (DIT).

A basic knowledge of the functionality of the **slapd−ldap**(5) backend is recommended. This backend has been designed as an enhancement of the ldap backend. The two backends share many features (actually they also share portions of code). While the **ldap** backend is intended to proxy operations directed to a single server, the **meta** backend is mainly intended for proxying of multiple servers and possibly naming context masquerading. These features, although useful in many scenarios, may result in excessive overhead for some applications, so its use should be carefully considered. In the examples section, some typical scenarios will be discussed.

Note: When looping back to the same instance of **slapd**(8), each connection requires a new thread; as a consequence, **slapd**(8) must be compiled with thread support, and the **threads** parameter may need some tuning; in those cases, unless the multiple target feature is required, one may consider using **slapd-relay**(5) instead, which performs the relayed operation internally and thus reuses the same connection.

EXAMPLES

There are examples in various places in this document, as well as in the slapd/back-meta/data/ directory in the OpenLDAP source tree.

CONFIGURATION

These **slapd.conf** options apply to the META backend database. That is, they must follow a "database meta" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

Note: In early versions of back-ldap and back-meta it was recommended to always set

```
lastmod off
```

for **ldap** and **meta** databases. This was required because operational attributes related to entry creation and modification should not be proxied, as they could be mistakenly written to the target server(s), generating an error. The current implementation automatically sets lastmod to **off**, so its use is redundant and should be omitted.

SPECIAL CONFIGURATION DIRECTIVES

Target configuration starts with the "uri" directive. All the configuration directives that are not specific to targets should be defined first for clarity, including those that are common to all backends. They are:

conn-ttl <time>

This directive causes a cached connection to be dropped an recreated after a given ttl, regardless of being idle or not.

default-target none

This directive forces the backend to reject all those operations that must resolve to a single target in case none or multiple targets are selected. They include: add, delete, modify, modrdn; compare is not included, as well as bind since, as they don’t alter entries, in case of multiple matches an attempt is made to perform the operation on any candidate target, with the constraint that at most one must succeed. This directive can also be used when processing targets to mark a specific target as default.

dncache-ttl {DISABLED|forever|<ttl>}
This directive sets the time-to-live of the DN cache. This caches the target that holds a given DN to speed up target selection in case multiple targets would result from an uncached search; forever means cache never expires; disabled means no DN caching; otherwise a valid (> 0) ttl is required, in the format illustrated for the **idle-timeout** directive.

onerr {CONTINUE|report|stop}
This directive allows to select the behavior in case an error is returned by one target during a search. The default, **continue**, consists in continuing the operation, trying to return as much data as possible. If the value is set to **stop**, the search is terminated as soon as an error is returned by one target, and the error is immediately propagated to the client. If the value is set to **report**, the search is continuated to the end but, in case at least one target returned an error code, the first non-success error code is returned.

protocol−version {0,2,3}
This directive indicates what protocol version must be used to contact the remote server. If set to 0 (the default), the proxy uses the same protocol version used by the client, otherwise the requested protocol is used. The proxy returns *unwillingToPerform* if an operation that is incompatible with the requested protocol is attempted. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

pseudoroot-bind-defer {YES|no}
This directive, when set to **yes**, causes the authentication to the remote servers with the pseudo-root identity to be deferred until actually needed by subsequent operations. Otherwise, all binds as the rootdn are propagated to the targets.

quarantine <interval>,<num>[;<interval>,<num>[...]]
Turns on quarantine of URIs that returned *LDAP_UNAVAILABLE*, so that an attempt to reconnect only occurs at given intervals instead of any time a client requests an operation. The pattern is: retry only after at least *interval* seconds elapsed since last attempt, for exactly *num* times; then use the next pattern. If *num* for the last pattern is "+", it retries forever; otherwise, no more retries occur. This directive must appear before any target specification; it affects all targets with the same pattern.

rebind-as-user {NO|yes}
If this option is given, the client's bind credentials are remembered for rebinds, when trying to re-establish a broken connection, or when chasing a referral, if **chase-referrals** is set to *yes*.

session−tracking−request {NO|yes}
Adds session tracking control for all requests. The client's IP and hostname, and the identity associated to each request, if known, are sent to the remote server for informational purposes. This directive is incompatible with setting *protocol−version* to 2. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

single−conn {NO|yes}
Discards current cached connection when the client rebinds.

use-temporary-conn {NO|yes}
when set to **yes**, create a temporary connection whenever competing with other threads for a shared one; otherwise, wait until the shared connection is available.

TARGET SPECIFICATION

Target specification starts with a "uri" directive:

uri <protocol>://[<host>]/<naming context> [...]

The <protocol> part can be anything **ldap_initialize**(3) accepts ({ldap|ldaps|ldapi} and variants); the <host> may be omitted, defaulting to whatever is set in **ldap.conf**(5). The <naming context> part is *mandatory* for the first URI, but it *must be omitted* for subsequent ones, if any. The naming context part must be within the naming context defined for the backend, e.g.:

suffix "**dc=foo,dc=com**"
uri "ldap://x.foo.com/dc=x,**dc=foo,dc=com**"

The <naming context> part doesn't need to be unique across the targets; it may also match one of the values of the "suffix" directive. Multiple URIs may be defined in a single URI statement. The additional URIs must be separate arguments and must not have any <naming context> part. This causes the underlying library to contact the first server of the list that responds. For example, if *l1.foo.com* and *l2.foo.com* are shadows of the same server, the directive

suffix "**dc=foo,dc=com**"
uri "ldap://l1.foo.com/**dc=foo,dc=com**" "ldap://l2.foo.com/"

causes *l2.foo.com* to be contacted whenever *l1.foo.com* does not respond. In that case, the URI list is internally rearranged, by moving unavailable URIs to the end, so that further connection attempts occur with respect to the last URI that succeeded.

acl-authcDN <administrative DN for access control purposes>

DN which is used to query the target server for acl checking, as in the LDAP backend; it is supposed to have read access on the target server to attributes used on the proxy for acl checking. There is no risk of giving away such values; they are only used to check permissions. **The acl-authcDN identity is by no means implicitly used by the proxy when the client connects anonymously.**

acl-passwd <password>

Password used with the **acl-authcDN** above.

bind-timeout <microseconds>

This directive defines the timeout, in microseconds, used when polling for response after an asynchronous bind connection. The initial call to ldap_result(3) is performed with a trade-off timeout of 100000 us; if that results in a timeout exceeded, subsequent calls use the value provided with **bind-timeout**. The default value is used also for subsequent calls if **bind-timeout** is not specified. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

chase-referrals {YES|no}

enable/disable automatic referral chasing, which is delegated to the underlying libldap, with rebinding eventually performed if the **rebind-as-user** directive is used. The default is to chase referrals. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

default-target [<target>]

The "default-target" directive can also be used during target specification. With no arguments it marks the current target as the default. The optional number marks target <target> as the default one, starting from 1. Target <target> must be defined.

idassert-authzFrom <authz-regexp>
if defined, selects what *local* identities are authorized to exploit the identity assertion feature. The string **<authz-regexp>** follows the rules defined for the *authzFrom* attribute. See **slapd.conf**(5), section related to **authz-policy**, for details on the syntax of this field.

idassert-bind bindmethod=none|simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [authz={native|proxyauthz}] [mode=<mode>] [flags=<flags>] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Allows to define the parameters of the authentication method that is internally used by the proxy to authorize connections that are authenticated by other databases. The identity defined by this directive, according to the properties associated to the authentication method, is supposed to have auth access on the target server to attributes used on the proxy for authentication and authorization, and to be allowed to authorize the users. This requires to have **proxyAuthz** privileges on a wide set of DNs, e.g. **authzTo=dn.subtree:""**, and the remote server to have **authz-policy** set to **to** or **both**. See **slapd.conf**(5) for details on these statements and for remarks and drawbacks about their usage. The supported bindmethods are

none|simple|sasl

where **none** is the default, i.e. no *identity assertion* is performed.

The authz parameter is used to instruct the SASL bind to exploit **native** SASL authorization, if available; since connections are cached, this should only be used when authorizing with a fixed identity (e.g. by means of the **authzDN** or **authzID** parameters). Otherwise, the default **proxyauthz** is used, i.e. the proxyAuthz control (Proxied Authorization, RFC 4370) is added to all operations.

The supported modes are:

<mode> := {legacy|anonymous|none|self}

If **<mode>** is not present, and **authzId** is given, the proxy always authorizes that identity. **<authorization ID>** can be

u:<user>

[dn:]<DN>

The former is supposed to be expanded by the remote server according to the authz rules; see **slapd.conf**(5) for details. In the latter case, whether or not the **dn:** prefix is present, the string must pass DN validation and normalization.

The default mode is **legacy**, which implies that the proxy will either perform a simple bind as the *authcDN* or a SASL bind as the *authcID* and assert the client's identity when it is not anonymous. Direct binds are always proxied. The other modes imply that the proxy will always either perform a simple bind as the *authcDN* or a SASL bind as the *authcID*, unless restricted by **idassert-authzFrom** rules (see below), in which case the operation will fail; eventually, it will assert some other identity according to **<mode>**. Other identity assertion modes are **anonymous** and **self**, which respectively mean that the *empty* or the *client*'s identity will be asserted; **none**, which means that no proxyAuthz control will be used, so the *authcDN* or the *authcID* identity will be asserted. For all modes that require the use of the *proxyAuthz* control, on the remote server the

proxy identity must have appropriate *authzTo* permissions, or the asserted identities must have appropriate *authzFrom* permissions. Note, however, that the ID assertion feature is mostly useful when the asserted identities do not exist on the remote server.

Flags can be

override,[non-]prescriptive

When the **override** flag is used, identity assertion takes place even when the database is authorizing for the identity of the client, i.e. after binding with the provided identity, and thus authenticating it, the proxy performs the identity assertion using the configured identity and authentication method.

When the **prescriptive** flag is used (the default), operations fail with *inappropriateAuthentication* for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns. If the **non-prescriptive** flag is used, operations are performed anonymously for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

The identity associated to this directive is also used for privileged operations whenever **idassert-bind** is defined and **acl-bind** is not. See **acl-bind** for details.

idle-timeout <time>

This directive causes a cached connection to be dropped an recreated after it has been idle for the specified time. The value can be specified as

[<d>d][<h>h][<m>m][<s>[s]]

where <d>, <h>, <m> and <s> are respectively treated as days, hours, minutes and seconds. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

map {attribute|objectclass} [<local name>|*] {<foreign name>|*}

This maps object classes and attributes as in the LDAP backend. See **slapd-ldap**(5).

network-timeout <time>

Sets the network timeout value after which **poll**(2)/**select**(2) following a **connect**(2) returns in case of no activity. The value is in seconds, and it can be specified as for **idle-timeout**. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

nretries {forever|never|<nretries>}

This directive defines how many times a bind should be retried in case of temporary failure in contacting a target. If defined before any target specification, it applies to all targets (by default, **3** times); the global value can be overridden by redefinitions inside each target specification.

pseudorootdn <substitute DN in case of rootdn bind>

This directive, if present, sets the DN that will be substituted to the bind DN if a bind with the backend's "rootdn" succeeds. The true "rootdn" of the target server ought not be used; an arbitrary administrative DN should used instead.

pseudorootpw <substitute password in case of rootdn bind>
This directive sets the credential that will be used in case a bind with the backend's "rootdn" succeeds, and the bind is propagated to the target using the "pseudorootdn" DN.

Note: cleartext credentials must be supplied here; as a consequence, using the pseudorootdn/pseudorootpw directives is inherently unsafe.

rewrite* ...
The rewrite options are described in the "REWRITING" section.

subtree-exclude <DN>
This directive instructs back-meta to ignore the current target for operations whose requestDN is subordinate to **DN**. There may be multiple occurrences of the **subtree-exclude** directive for each of the targets.

suffixmassage <virtual naming context> <real naming context>
All the directives starting with "rewrite" refer to the rewrite engine that has been added to slapd. The "suffixmassage" directive was introduced in the LDAP backend to allow suffix massaging while proxying. It has been obsoleted by the rewriting tools. However, both for backward compatibility and for ease of configuration when simple suffix massage is required, it has been preserved. It wraps the basic rewriting instructions that perform suffix massaging. See the "REWRITING" section for a detailed list of the rewrite rules it implies.

t-f-support {NO|yes|discover}
enable if the remote server supports absolute filters (see *draft-zeilenga-ldap-t-f* for details). If set to **discover**, support is detected by reading the remote server's root DSE. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

timeout [<op>=]<val> [...]
This directive allows to set per-operation timeouts. Operations can be

<op> ::= bind, add, delete, modrdn, modify, compare, search

The overall duration of the **search** operation is controlled either by the **timelimit** parameter or by server-side enforced time limits (see **timelimit** and **limits** in **slapd.conf**(5) for details). This **timeout** parameter controls how long the target can be irresponsive before the operation is aborted. Timeout is meaningless for the remaining operations, **unbind** and **abandon**, which do not imply any response, while it is not yet implemented in currently supported **extended** operations. If no operation is specified, the timeout **val** affects all supported operations. If specified before any target definition, it affects all targets unless overridden by per-target directives.

Note: if the timeout is exceeded, the operation is cancelled (according to the **cancel** directive); the protocol does not provide any means to rollback operations, so the client will not be notified about the result of the operation, which may eventually succeeded or not. In case the timeout is exceeded during a bind operation, the connection is destroyed, according to RFC4511.

tls {[try-]start|[try-]propagate}
execute the StartTLS extended operation when the connection is initialized; only works if the URI directive protocol scheme is not **ldaps://**. **propagate** issues the StartTLS operation only if the original connection did. The **try-** prefix instructs the proxy to continue operations if the StartTLS operation failed; its use is highly deprecated. If set before any target specification, it affects all

targets, unless overridden by any per-target directive.

SCENARIOS

A powerful (and in some sense dangerous) rewrite engine has been added to both the LDAP and Meta backends. While the former can gain limited beneficial effects from rewriting stuff, the latter can become an amazingly powerful tool.

Consider a couple of scenarios first.

1) Two directory servers share two levels of naming context; say "dc=a,dc=foo,dc=com" and "dc=b,dc=foo,dc=com". Then, an unambiguous Meta database can be configured as:

```
database meta
suffix   "dc=foo,dc=com"
uri      "ldap://a.foo.com/dc=a,dc=foo,dc=com"
uri      "ldap://b.foo.com/dc=b,dc=foo,dc=com"
```

Operations directed to a specific target can be easily resolved because there are no ambiguities. The only operation that may resolve to multiple targets is a search with base "dc=foo,dc=com" and scope at least "one", which results in spawning two searches to the targets.

2a) Two directory servers don't share any portion of naming context, but they'd present as a single DIT [Caveat: uniqueness of (massaged) entries among the two servers is assumed; integrity checks risk to incur in excessive overhead and have not been implemented]. Say we have "dc=bar,dc=org" and "o=Foo,c=US", and we'd like them to appear as branches of "dc=foo,dc=com", say "dc=a,dc=foo,dc=com" and "dc=b,dc=foo,dc=com". Then we need to configure our Meta backend as:

```
database     meta
suffix       "dc=foo,dc=com"

uri          "ldap://a.bar.com/dc=a,dc=foo,dc=com"
suffixmassage "dc=a,dc=foo,dc=com" "dc=bar,dc=org"

uri          "ldap://b.foo.com/dc=b,dc=foo,dc=com"
suffixmassage "dc=b,dc=foo,dc=com" "o=Foo,c=US"
```

Again, operations can be resolved without ambiguity, although some rewriting is required. Notice that the virtual naming context of each target is a branch of the database's naming context; it is rewritten back and forth when operations are performed towards the target servers. What "back and forth" means will be clarified later.

When a search with base "dc=foo,dc=com" is attempted, if the scope is "base" it fails with "no such object"; in fact, the common root of the two targets (prior to massaging) does not exist. If the scope is "one", both targets are contacted with the base replaced by each target's base; the scope is derated to "base". In general, a scope "one" search is honored, and the scope is derated, only when the incoming base is at most one level lower of a target's naming context (prior to massaging).

Finally, if the scope is "sub" the incoming base is replaced by each target's unmassaged naming context, and the scope is not altered.

2b) Consider the above reported scenario with the two servers sharing the same naming context:

```
database     meta
suffix       "dc=foo,dc=com"

uri          "ldap://a.bar.com/dc=foo,dc=com"
suffixmassage "dc=foo,dc=com" "dc=bar,dc=org"

uri          "ldap://b.foo.com/dc=foo,dc=com"
suffixmassage "dc=foo,dc=com" "o=Foo,c=US"
```

All the previous considerations hold, except that now there is no way to unambiguously resolve a DN. In this case, all the operations that require an unambiguous target selection will fail unless the DN is already cached or a default target has been set. Practical configurations may result as a combination of all the above scenarios.

ACLs

Note on ACLs: at present you may add whatever ACL rule you desire to to the Meta (and LDAP) backends. However, the meaning of an ACL on a proxy may require some considerations. Two philosophies may be considered:

a) the remote server dictates the permissions; the proxy simply passes back what it gets from the remote server.

b) the remote server unveils "everything"; the proxy is responsible for protecting data from unauthorized access.

Of course the latter sounds unreasonable, but it is not. It is possible to imagine scenarios in which a remote host discloses data that can be considered "public" inside an intranet, and a proxy that connects it to the internet may impose additional constraints. To this purpose, the proxy should be able to comply with all the ACL matching criteria that the server supports. This has been achieved with regard to all the criteria supported by slapd except a special subtle case (please drop me a note if you can find other exceptions: <ando@openldap.org>). The rule

```
access to dn="<dn>" attrs=<attr>
    by dnattr=<dnattr> read
    by * none
```

cannot be matched iff the attribute that is being requested, <attr>, is NOT <dnattr>, and the attribute that determines membership, <dnattr>, has not been requested (e.g. in a search)

In fact this ACL is resolved by slapd using the portion of entry it retrieved from the remote server without requiring any further intervention of the backend, so, if the <dnattr> attribute has not been fetched, the match cannot be assessed because the attribute is not present, not because no value matches the requirement!

Note on ACLs and attribute mapping: ACLs are applied to the mapped attributes; for instance, if the attribute locally known as "foo" is mapped to "bar" on a remote server, then local ACLs apply to attribute "foo" and are totally unaware of its remote name. The remote server will check permissions for "bar", and the local server will possibly enforce additional restrictions to "foo".

REWRITING

A string is rewritten according to a set of rules, called a ‘rewrite context’. The rules are based on POSIX (’’extended’’) regular expressions (regex) with substring matching; basic variable substitution and map resolution of substrings is allowed by specific mechanisms detailed in the following. The behavior of pattern matching/substitution can be altered by a set of flags.

The underlying concept is to build a lightweight rewrite module for the slapd server (initially dedicated to the LDAP backend).

Passes

An incoming string is matched against a set of rules. Rules are made of a regex match pattern, a substitution pattern and a set of actions, described by a set of flags. In case of match a string rewriting is performed according to the substitution pattern that allows to refer to substrings matched in the incoming string. The actions, if any, are finally performed. The substitution pattern allows map resolution of substrings. A map is a generic object that maps a substitution pattern to a value. The flags are divided in "Pattern matching Flags" and "Action Flags"; the former alter the regex match pattern behavior while the latter alter the action that is taken after substitution.

Pattern Matching Flags

‘C’ honors case in matching (default is case insensitive)

‘R’ use POSIX ’’basic’’ regular expressions (default is ’’extended’’)

‘M{n}’ allow no more than **n** recursive passes for a specific rule; does not alter the max total count of passes, so it can only enforce a stricter limit for a specific rule.

Action Flags

‘:’ apply the rule once only (default is recursive)

‘@’ stop applying rules in case of match; the current rule is still applied recursively; combine with ‘:’ to apply the current rule only once and then stop.

‘#’ stop current operation if the rule matches, and issue an ‘unwilling to perform’ error.

‘G{n}’ jump **n** rules back and forth (watch for loops!). Note that ‘G{1}’ is implicit in every rule.

‘I’ ignores errors in rule; this means, in case of error, e.g. issued by a map, the error is treated as a missed match. The ‘unwilling to perform’ is not overridden.

‘U{n}’ uses **n** as return code if the rule matches; the flag does not alter the recursive behavior of the rule, so, to have it performed only once, it must be used in combination with ‘:’, e.g. **‘:U{16}’** returns the value ‘16’ after exactly one execution of the rule, if the pattern matches. As a consequence, its behavior is equivalent to ‘@’, with the return code set to **n**; or, in other words, ‘@’ is equivalent to ‘U{0}’. By convention, the freely available codes are above 16 included; the others are reserved.

The ordering of the flags can be significant. For instance: ‘IG{2}’ means ignore errors and jump two lines ahead both in case of match and in case of error, while ‘G{2}I’ means ignore errors, but jump two lines ahead only in case of match.

More flags (mainly Action Flags) will be added as needed.

Pattern matching:

See **regex**(7) and/or **re_format**(7).

Substitution Pattern Syntax:

Everything starting with ‘%’ requires substitution;

the only obvious exception is ‘%%’, which is left as is;

the basic substitution is ‘%d’, where ‘d’ is a digit; 0 means the whole string, while 1-9 is a submatch;

a ‘%’ followed by a ‘{’ invokes an advanced substitution. The pattern is:

‘%’ ‘{’ [<op>] <name> ‘(’ <substitution> ‘)’ ‘}’

where <name> must be a legal name for the map, i.e.

<name> ::= [a-z][a-z0-9]* (case insensitive)
<op> ::= ‘>’ ‘|’ ‘&’ ‘&&’ ‘*’ ‘**’ ‘$’

and <substitution> must be a legal substitution pattern, with no limits on the nesting level.

The operators are:

> sub context invocation; <name> must be a legal, already defined rewrite context name

| external command invocation; <name> must refer to a legal, already defined command name (NOT IMPL.)

& variable assignment; <name> defines a variable in the running operation structure which can be dereferenced later; operator **&** assigns a variable in the rewrite context scope; operator **&&** assigns a variable that scopes the entire session, e.g. its value can be dereferenced later by other rewrite contexts

***** variable dereferencing; <name> must refer to a variable that is defined and assigned for the running operation; operator ***** dereferences a variable scoping the rewrite context; operator ****** dereferences a variable scoping the whole session, e.g. the value is passed across rewrite contexts

$ parameter dereferencing; <name> must refer to an existing parameter; the idea is to make some run-time parameters set by the system available to the rewrite engine, as the client host name, the

bind DN if any, constant parameters initialized at config time, and so on; no parameter is currently set by either **back−ldap** or **back−meta**, but constant parameters can be defined in the configuration file by using the **rewriteParam** directive.

Substitution escaping has been delegated to the '%' symbol, which is used instead of '\' in string substitution patterns because '\' is already escaped by slapd's low level parsing routines; as a consequence, regex escaping requires two '\' symbols, e.g. '**.*\.foo\.bar**' must be written as '**.*\\.foo\\.bar**'.

Rewrite context:

A rewrite context is a set of rules which are applied in sequence. The basic idea is to have an application initialize a rewrite engine (think of Apache's mod_rewrite ...) with a set of rewrite contexts; when string rewriting is required, one invokes the appropriate rewrite context with the input string and obtains the newly rewritten one if no errors occur.

Each basic server operation is associated to a rewrite context; they are divided in two main groups: client −> server and server −> client rewriting.

client -> server:

(default)	if defined and no specific context is available
bindDN	bind
searchBase	search
searchFilter	search
searchFilterAttrDN	search
compareDN	compare
compareAttrDN	compare AVA
addDN	add
addAttrDN	add AVA
modifyDN	modify
modifyAttrDN	modify AVA
modrDN	modrdn
newSuperiorDN	modrdn
deleteDN	delete
exopPasswdDN	password modify extended operation DN if proxy

server -> client:

searchResult	search (only if defined; no default; acts on DN and DN-syntax attributes of search results)
searchAttrDN	search AVA
matchedDN	all ops (only if applicable)

Basic configuration syntax

rewriteEngine { on | off }

If 'on', the requested rewriting is performed; if 'off', no rewriting takes place (an easy way to stop rewriting without altering too much the configuration file).

rewriteContext <context name> [alias <aliased context name>]

<Context name> is the name that identifies the context, i.e. the name used by the application to refer to the set of rules it contains. It is used also to reference sub contexts in string rewriting. A context may alias another one. In this case the alias context contains no rule, and any reference to it will result in accessing the aliased one.

rewriteRule <regex match pattern> <substitution pattern> [<flags>]

Determines how a string can be rewritten if a pattern is matched. Examples are reported below.

Additional configuration syntax:

rewriteMap <map type> <map name> [<map attrs>]
Allows to define a map that transforms substring rewriting into something else. The map is referenced inside the substitution pattern of a rule.

rewriteParam <param name> <param value>
Sets a value with global scope, that can be dereferenced by the command '%{$paramName}'.

rewriteMaxPasses <number of passes> [<number of passes per rule>]
Sets the maximum number of total rewriting passes that can be performed in a single rewrite operation (to avoid loops). A safe default is set to 100; note that reaching this limit is still treated as a success; recursive invocation of rules is simply interrupted. The count applies to the rewriting operation as a whole, not to any single rule; an optional per-rule limit can be set. This limit is overridden by setting specific per-rule limits with the 'M{n}' flag.

Configuration examples:

```
# set to 'off' to disable rewriting
rewriteEngine on

# the rules the "suffixmassage" directive implies
rewriteEngine on
# all dataflow from client to server referring to DNs
rewriteContext default
rewriteRule "(.*)<virtualnamingcontext>$" "%1<realnamingcontext>" ":"
# empty filter rule
rewriteContext searchFilter
# all dataflow from server to client
rewriteContext searchResult
rewriteRule "(.*)<realnamingcontext>$" "%1<virtualnamingcontext>" ":"
rewriteContext searchAttrDN alias searchResult
rewriteContext matchedDN alias searchResult

# Everything defined here goes into the 'default' context.
# This rule changes the naming context of anything sent
# to 'dc=home,dc=net' to 'dc=OpenLDAP, dc=org'

rewriteRule "(.*)dc=home,[ ]?dc=net"
        "%1dc=OpenLDAP, dc=org"  ":"

# since a pretty/normalized DN does not include spaces
# after rdn separators, e.g. ',', this rule suffices:

rewriteRule "(.*)dc=home,dc=net"
        "%1dc=OpenLDAP,dc=org"  ":"

# Start a new context (ends input of the previous one).
# This rule adds blanks between DN parts if not present.
rewriteContext  addBlanks
rewriteRule     "(.*),([^ ].*)" "%1, %2"

# This one eats blanks
rewriteContext  eatBlanks
rewriteRule     "(.*),[ ](.*)" "%1,%2"

# Here control goes back to the default rewrite
# context; rules are appended to the existing ones.
# anything that gets here is piped into rule 'addBlanks'
```

```
rewriteContext  default
rewriteRule     ".*" "%{>addBlanks(%0)}" ":"

# Rewrite the search base according to ‘default’ rules.
rewriteContext  searchBase alias default

# Search results with OpenLDAP DN are rewritten back with
# ‘dc=home,dc=net’ naming context, with spaces eaten.
rewriteContext  searchResult
rewriteRule     "(.*[^ ]?)[ ]?dc=OpenLDAP,[ ]?dc=org"
                "%{>eatBlanks(%1)}dc=home,dc=net"    ":"

# Bind with email instead of full DN: we first need
# an ldap map that turns attributes into a DN (the
# argument used when invoking the map is appended to
# the URI and acts as the filter portion)
rewriteMap ldap attr2dn "ldap://host/dc=my,dc=org?dn?sub"

# Then we need to detect DN made up of a single email,
# e.g. ‘mail=someone@example.com’; note that the rule
# in case of match stops rewriting; in case of error,
# it is ignored.  In case we are mapping virtual
# to real naming contexts, we also need to rewrite
# regular DNs, because the definition of a bindDn
# rewrite context overrides the default definition.
rewriteContext bindDN
rewriteRule "^mail=[^,]+@[^,]+$" "%{attr2dn(%0)}" ":@I"

# This is a rather sophisticated example. It massages a
# search filter in case who performs the search has
# administrative privileges.  First we need to keep
# track of the bind DN of the incoming request, which is
# stored in a variable called ‘binddn’ with session scope,
# and left in place to allow regular binding:
rewriteContext  bindDN
rewriteRule     ".+" "%{&&binddn(%0)}%0" ":"

# A search filter containing ‘uid=’ is rewritten only
# if an appropriate DN is bound.
# To do this, in the first rule the bound DN is
# dereferenced, while the filter is decomposed in a
# prefix, in the value of the ‘uid=<arg>’ AVA, and
# in a suffix. A tag ‘<>’ is appended to the DN.
# If the DN refers to an entry in the ‘ou=admin’ subtree,
# the filter is rewritten OR-ing the ‘uid=<arg>’ with
# ‘cn=<arg>’; otherwise it is left as is. This could be
# useful, for instance, to allow apache’s auth_ldap-1.4
# module to authenticate users with both ‘uid’ and
# ‘cn’, but only if the request comes from a possible
# ‘cn=Web auth,ou=admin,dc=home,dc=net’ user.
rewriteContext searchFilter
rewriteRule "(.*\\()uid=([a-z0-9_]+)(\\).*)"
  "%{**binddn}<>%{&prefix(%1)}%{&arg(%2)}%{&suffix(%3)}"
  ":I"
```

```
rewriteRule "[^,]+,ou=admin,dc=home,dc=net"
  "%{*prefix}|(uid=%{*arg})(cn=%{*arg})%{*suffix}" ":@I"
rewriteRule ".*<>" "%{*prefix}uid=%{*arg}%{*suffix}" ":"

# This example shows how to strip unwanted DN-valued
# attribute values from a search result; the first rule
# matches DN values below "ou=People,dc=example,dc=com";
# in case of match the rewriting exits successfully.
# The second rule matches everything else and causes
# the value to be rejected.
rewriteContext searchResult
rewriteRule ".*,ou=People,dc=example,dc=com" "%0" ":@"
rewriteRule ".*" "" "#"
```

LDAP Proxy resolution (a possible evolution of slapd−ldap(5)):

In case the rewritten DN is an LDAP URI, the operation is initiated towards the host[:port] indicated in the uri, if it does not refer to the local server. E.g.:

```
rewriteRule '^cn=root,.*' '%0'                  'G{3}'
rewriteRule '^cn=[a-l].*' 'ldap://ldap1.my.org/%0' ':@'
rewriteRule '^cn=[m-z].*' 'ldap://ldap2.my.org/%0' ':@'
rewriteRule '.*'          'ldap://ldap3.my.org/%0' ':@'
```

(Rule 1 is simply there to illustrate the 'G{n}' action; it could have been written:

```
rewriteRule '^cn=root,.*' 'ldap://ldap3.my.org/%0' ':@'
```

with the advantage of saving one rewrite pass ...)

ACCESS CONTROL

The **meta** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access checking is delegated to the remote server(s). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

PROXY CACHE OVERLAY

The proxy cache overlay allows caching of LDAP search requests (queries) in a local database. See **slapo-pcache**(5) for details.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapo−pcache**(5), **slapd**(8), **regex**(7), **re_format**(7).

AUTHOR

Pierangelo Masarati, based on back-ldap by Howard Chu

NAME

slapd-meta − metadirectory backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **meta** backend to **slapd**(8) performs basic LDAP proxying with respect to a set of remote LDAP servers, called "targets". The information contained in these servers can be presented as belonging to a single Directory Information Tree (DIT).

A basic knowledge of the functionality of the **slapd−ldap**(5) backend is recommended. This backend has been designed as an enhancement of the ldap backend. The two backends share many features (actually they also share portions of code). While the **ldap** backend is intended to proxy operations directed to a single server, the **meta** backend is mainly intended for proxying of multiple servers and possibly naming context masquerading. These features, although useful in many scenarios, may result in excessive overhead for some applications, so its use should be carefully considered. In the examples section, some typical scenarios will be discussed.

Note: When looping back to the same instance of **slapd**(8), each connection requires a new thread; as a consequence, **slapd**(8) must be compiled with thread support, and the **threads** parameter may need some tuning; in those cases, unless the multiple target feature is required, one may consider using **slapd-relay**(5) instead, which performs the relayed operation internally and thus reuses the same connection.

EXAMPLES

There are examples in various places in this document, as well as in the slapd/back-meta/data/ directory in the OpenLDAP source tree.

CONFIGURATION

These **slapd.conf** options apply to the META backend database. That is, they must follow a "database meta" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

Note: In early versions of back-ldap and back-meta it was recommended to always set

```
lastmod  off
```

for **ldap** and **meta** databases. This was required because operational attributes related to entry creation and modification should not be proxied, as they could be mistakenly written to the target server(s), generating an error. The current implementation automatically sets lastmod to **off**, so its use is redundant and should be omitted.

SPECIAL CONFIGURATION DIRECTIVES

Target configuration starts with the "uri" directive. All the configuration directives that are not specific to targets should be defined first for clarity, including those that are common to all backends. They are:

conn-ttl <time>

This directive causes a cached connection to be dropped an recreated after a given ttl, regardless of being idle or not.

default-target none

This directive forces the backend to reject all those operations that must resolve to a single target in case none or multiple targets are selected. They include: add, delete, modify, modrdn; compare is not included, as well as bind since, as they don’t alter entries, in case of multiple matches an attempt is made to perform the operation on any candidate target, with the constraint that at most one must succeed. This directive can also be used when processing targets to mark a specific target as default.

dncache-ttl {DISABLED|forever|<ttl>}
This directive sets the time-to-live of the DN cache. This caches the target that holds a given DN to speed up target selection in case multiple targets would result from an uncached search; forever means cache never expires; disabled means no DN caching; otherwise a valid (> 0) ttl is required, in the format illustrated for the **idle-timeout** directive.

onerr {CONTINUE|report|stop}
This directive allows to select the behavior in case an error is returned by one target during a search. The default, **continue**, consists in continuing the operation, trying to return as much data as possible. If the value is set to **stop**, the search is terminated as soon as an error is returned by one target, and the error is immediately propagated to the client. If the value is set to **report**, the search is continuated to the end but, in case at least one target returned an error code, the first non-success error code is returned.

protocol−version {0,2,3}
This directive indicates what protocol version must be used to contact the remote server. If set to 0 (the default), the proxy uses the same protocol version used by the client, otherwise the requested protocol is used. The proxy returns *unwillingToPerform* if an operation that is incompatible with the requested protocol is attempted. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

pseudoroot-bind-defer {YES|no}
This directive, when set to **yes**, causes the authentication to the remote servers with the pseudo-root identity to be deferred until actually needed by subsequent operations. Otherwise, all binds as the rootdn are propagated to the targets.

quarantine <interval>,<num>[;<interval>,<num>[...]]
Turns on quarantine of URIs that returned *LDAP_UNAVAILABLE*, so that an attempt to reconnect only occurs at given intervals instead of any time a client requests an operation. The pattern is: retry only after at least *interval* seconds elapsed since last attempt, for exactly *num* times; then use the next pattern. If *num* for the last pattern is "+", it retries forever; otherwise, no more retries occur. This directive must appear before any target specification; it affects all targets with the same pattern.

rebind-as-user {NO|yes}
If this option is given, the client's bind credentials are remembered for rebinds, when trying to re-establish a broken connection, or when chasing a referral, if **chase-referrals** is set to *yes*.

session−tracking−request {NO|yes}
Adds session tracking control for all requests. The client's IP and hostname, and the identity associated to each request, if known, are sent to the remote server for informational purposes. This directive is incompatible with setting *protocol−version* to 2. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

single−conn {NO|yes}
Discards current cached connection when the client rebinds.

use-temporary-conn {NO|yes}
when set to **yes**, create a temporary connection whenever competing with other threads for a shared one; otherwise, wait until the shared connection is available.

TARGET SPECIFICATION

Target specification starts with a "uri" directive:

uri <protocol>://[<host>]/<naming context> [...]

The <protocol> part can be anything **ldap_initialize**(3) accepts ({ldap|ldaps|ldapi} and variants); the <host> may be omitted, defaulting to whatever is set in **ldap.conf**(5). The <naming context> part is *mandatory* for the first URI, but it *must be omitted* for subsequent ones, if any. The naming context part must be within the naming context defined for the backend, e.g.:

suffix "**dc=foo,dc=com**"
uri "ldap://x.foo.com/dc=x,**dc=foo,dc=com**"

The <naming context> part doesn't need to be unique across the targets; it may also match one of the values of the "suffix" directive. Multiple URIs may be defined in a single URI statement. The additional URIs must be separate arguments and must not have any <naming context> part. This causes the underlying library to contact the first server of the list that responds. For example, if *l1.foo.com* and *l2.foo.com* are shadows of the same server, the directive

suffix "**dc=foo,dc=com**"
uri "ldap://l1.foo.com/**dc=foo,dc=com**" "ldap://l2.foo.com/"

causes *l2.foo.com* to be contacted whenever *l1.foo.com* does not respond. In that case, the URI list is internally rearranged, by moving unavailable URIs to the end, so that further connection attempts occur with respect to the last URI that succeeded.

acl-authcDN <administrative DN for access control purposes>

DN which is used to query the target server for acl checking, as in the LDAP backend; it is supposed to have read access on the target server to attributes used on the proxy for acl checking. There is no risk of giving away such values; they are only used to check permissions. **The acl-authcDN identity is by no means implicitly used by the proxy when the client connects anonymously.**

acl-passwd <password>

Password used with the **acl-authcDN** above.

bind-timeout <microseconds>

This directive defines the timeout, in microseconds, used when polling for response after an asynchronous bind connection. The initial call to ldap_result(3) is performed with a trade-off timeout of 100000 us; if that results in a timeout exceeded, subsequent calls use the value provided with **bind-timeout**. The default value is used also for subsequent calls if **bind-timeout** is not specified. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

chase-referrals {YES|no}

enable/disable automatic referral chasing, which is delegated to the underlying libldap, with rebinding eventually performed if the **rebind-as-user** directive is used. The default is to chase referrals. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

default-target [<target>]

The "default-target" directive can also be used during target specification. With no arguments it marks the current target as the default. The optional number marks target <target> as the default one, starting from 1. Target <target> must be defined.

idassert-authzFrom <authz-regexp>
if defined, selects what *local* identities are authorized to exploit the identity assertion feature. The string **<authz-regexp>** follows the rules defined for the *authzFrom* attribute. See **slapd.conf**(5), section related to **authz-policy**, for details on the syntax of this field.

idassert-bind bindmethod=none|simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [authz={native|proxyauthz}] [mode=<mode>] [flags=<flags>] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all]
Allows to define the parameters of the authentication method that is internally used by the proxy to authorize connections that are authenticated by other databases. The identity defined by this directive, according to the properties associated to the authentication method, is supposed to have auth access on the target server to attributes used on the proxy for authentication and authorization, and to be allowed to authorize the users. This requires to have **proxyAuthz** privileges on a wide set of DNs, e.g. **authzTo=dn.subtree:""**, and the remote server to have **authz-policy** set to **to** or **both**. See **slapd.conf**(5) for details on these statements and for remarks and drawbacks about their usage. The supported bindmethods are

none|simple|sasl

where **none** is the default, i.e. no *identity assertion* is performed.

The authz parameter is used to instruct the SASL bind to exploit **native** SASL authorization, if available; since connections are cached, this should only be used when authorizing with a fixed identity (e.g. by means of the **authzDN** or **authzID** parameters). Otherwise, the default **proxyauthz** is used, i.e. the proxyAuthz control (Proxied Authorization, RFC 4370) is added to all operations.

The supported modes are:

<mode> := {legacy|anonymous|none|self}

If **<mode>** is not present, and **authzId** is given, the proxy always authorizes that identity. **<authorization ID>** can be

u:<user>

[dn:]<DN>

The former is supposed to be expanded by the remote server according to the authz rules; see **slapd.conf**(5) for details. In the latter case, whether or not the **dn:** prefix is present, the string must pass DN validation and normalization.

The default mode is **legacy**, which implies that the proxy will either perform a simple bind as the *authcDN* or a SASL bind as the *authcID* and assert the client's identity when it is not anonymous. Direct binds are always proxied. The other modes imply that the proxy will always either perform a simple bind as the *authcDN* or a SASL bind as the *authcID*, unless restricted by **idassert-authzFrom** rules (see below), in which case the operation will fail; eventually, it will assert some other identity according to **<mode>**. Other identity assertion modes are **anonymous** and **self**, which respectively mean that the *empty* or the *client*'s identity will be asserted; **none**, which means that no proxyAuthz control will be used, so the *authcDN* or the *authcID* identity will be asserted. For all modes that require the use of the *proxyAuthz* control, on the remote server the

proxy identity must have appropriate *authzTo* permissions, or the asserted identities must have appropriate *authzFrom* permissions. Note, however, that the ID assertion feature is mostly useful when the asserted identities do not exist on the remote server.

Flags can be

override,[non-]prescriptive

When the **override** flag is used, identity assertion takes place even when the database is authorizing for the identity of the client, i.e. after binding with the provided identity, and thus authenticating it, the proxy performs the identity assertion using the configured identity and authentication method.

When the **prescriptive** flag is used (the default), operations fail with *inappropriateAuthentication* for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns. If the **non-prescriptive** flag is used, operations are performed anonymously for those identities whose assertion is not allowed by the **idassert-authzFrom** patterns.

The TLS settings default to the same as the main slapd TLS settings, except for **tls_reqcert** which defaults to "demand".

The identity associated to this directive is also used for privileged operations whenever **idassert-bind** is defined and **acl-bind** is not. See **acl-bind** for details.

idle-timeout <time>

This directive causes a cached connection to be dropped an recreated after it has been idle for the specified time. The value can be specified as

[<d>d][<h>h][<m>m][<s>[s]]

where <d>, <h>, <m> and <s> are respectively treated as days, hours, minutes and seconds. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

map {attribute|objectclass} [<local name>|*] {<foreign name>|*}

This maps object classes and attributes as in the LDAP backend. See **slapd-ldap**(5).

network-timeout <time>

Sets the network timeout value after which **poll**(2)/**select**(2) following a **connect**(2) returns in case of no activity. The value is in seconds, and it can be specified as for **idle-timeout**. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

nretries {forever|never|<nretries>}

This directive defines how many times a bind should be retried in case of temporary failure in contacting a target. If defined before any target specification, it applies to all targets (by default, **3** times); the global value can be overridden by redefinitions inside each target specification.

pseudorootdn <substitute DN in case of rootdn bind>

This directive, if present, sets the DN that will be substituted to the bind DN if a bind with the backend's "rootdn" succeeds. The true "rootdn" of the target server ought not be used; an arbitrary administrative DN should used instead.

pseudorootpw <substitute password in case of rootdn bind>
This directive sets the credential that will be used in case a bind with the backend's "rootdn" succeeds, and the bind is propagated to the target using the "pseudorootdn" DN.

Note: cleartext credentials must be supplied here; as a consequence, using the pseudorootdn/pseudorootpw directives is inherently unsafe.

rewrite* ...
The rewrite options are described in the "REWRITING" section.

subtree-exclude <DN>
This directive instructs back-meta to ignore the current target for operations whose requestDN is subordinate to **DN**. There may be multiple occurrences of the **subtree-exclude** directive for each of the targets.

suffixmassage <virtual naming context> <real naming context>
All the directives starting with "rewrite" refer to the rewrite engine that has been added to slapd. The "suffixmassage" directive was introduced in the LDAP backend to allow suffix massaging while proxying. It has been obsoleted by the rewriting tools. However, both for backward compatibility and for ease of configuration when simple suffix massage is required, it has been preserved. It wraps the basic rewriting instructions that perform suffix massaging. See the "REWRITING" section for a detailed list of the rewrite rules it implies.

t-f-support {NO|yes|discover}
enable if the remote server supports absolute filters (see *draft-zeilenga-ldap-t-f* for details). If set to **discover**, support is detected by reading the remote server's root DSE. If set before any target specification, it affects all targets, unless overridden by any per-target directive.

timeout [<op>=]<val> [...]
This directive allows to set per-operation timeouts. Operations can be

<op> ::= bind, add, delete, modrdn, modify, compare, search

The overall duration of the **search** operation is controlled either by the **timelimit** parameter or by server-side enforced time limits (see **timelimit** and **limits** in **slapd.conf**(5) for details). This **timeout** parameter controls how long the target can be irresponsive before the operation is aborted. Timeout is meaningless for the remaining operations, **unbind** and **abandon**, which do not imply any response, while it is not yet implemented in currently supported **extended** operations. If no operation is specified, the timeout **val** affects all supported operations. If specified before any target definition, it affects all targets unless overridden by per-target directives.

Note: if the timeout is exceeded, the operation is cancelled (according to the **cancel** directive); the protocol does not provide any means to rollback operations, so the client will not be notified about the result of the operation, which may eventually succeeded or not. In case the timeout is exceeded during a bind operation, the connection is destroyed, according to RFC4511.

tls {[try-]start|[try-]propagate}
execute the StartTLS extended operation when the connection is initialized; only works if the URI directive protocol scheme is not **ldaps://**. **propagate** issues the StartTLS operation only if the original connection did. The **try-** prefix instructs the proxy to continue operations if the StartTLS operation failed; its use is highly deprecated. If set before any target specification, it affects all

targets, unless overridden by any per-target directive.

SCENARIOS

A powerful (and in some sense dangerous) rewrite engine has been added to both the LDAP and Meta backends. While the former can gain limited beneficial effects from rewriting stuff, the latter can become an amazingly powerful tool.

Consider a couple of scenarios first.

1) Two directory servers share two levels of naming context; say "dc=a,dc=foo,dc=com" and "dc=b,dc=foo,dc=com". Then, an unambiguous Meta database can be configured as:

```
database meta
suffix   "dc=foo,dc=com"
uri      "ldap://a.foo.com/dc=a,dc=foo,dc=com"
uri      "ldap://b.foo.com/dc=b,dc=foo,dc=com"
```

Operations directed to a specific target can be easily resolved because there are no ambiguities. The only operation that may resolve to multiple targets is a search with base "dc=foo,dc=com" and scope at least "one", which results in spawning two searches to the targets.

2a) Two directory servers don't share any portion of naming context, but they'd present as a single DIT [Caveat: uniqueness of (massaged) entries among the two servers is assumed; integrity checks risk to incur in excessive overhead and have not been implemented]. Say we have "dc=bar,dc=org" and "o=Foo,c=US", and we'd like them to appear as branches of "dc=foo,dc=com", say "dc=a,dc=foo,dc=com" and "dc=b,dc=foo,dc=com". Then we need to configure our Meta backend as:

```
database     meta
suffix       "dc=foo,dc=com"

uri          "ldap://a.bar.com/dc=a,dc=foo,dc=com"
suffixmassage "dc=a,dc=foo,dc=com" "dc=bar,dc=org"

uri          "ldap://b.foo.com/dc=b,dc=foo,dc=com"
suffixmassage "dc=b,dc=foo,dc=com" "o=Foo,c=US"
```

Again, operations can be resolved without ambiguity, although some rewriting is required. Notice that the virtual naming context of each target is a branch of the database's naming context; it is rewritten back and forth when operations are performed towards the target servers. What "back and forth" means will be clarified later.

When a search with base "dc=foo,dc=com" is attempted, if the scope is "base" it fails with "no such object"; in fact, the common root of the two targets (prior to massaging) does not exist. If the scope is "one", both targets are contacted with the base replaced by each target's base; the scope is derated to "base". In general, a scope "one" search is honored, and the scope is derated, only when the incoming base is at most one level lower of a target's naming context (prior to massaging).

Finally, if the scope is "sub" the incoming base is replaced by each target's unmassaged naming context, and the scope is not altered.

2b) Consider the above reported scenario with the two servers sharing the same naming context:

```
database     meta
suffix       "dc=foo,dc=com"

uri          "ldap://a.bar.com/dc=foo,dc=com"
suffixmassage "dc=foo,dc=com" "dc=bar,dc=org"

uri          "ldap://b.foo.com/dc=foo,dc=com"
suffixmassage "dc=foo,dc=com" "o=Foo,c=US"
```

All the previous considerations hold, except that now there is no way to unambiguously resolve a DN. In this case, all the operations that require an unambiguous target selection will fail unless the DN is already cached or a default target has been set. Practical configurations may result as a combination of all the above scenarios.

ACLs

Note on ACLs: at present you may add whatever ACL rule you desire to to the Meta (and LDAP) backends. However, the meaning of an ACL on a proxy may require some considerations. Two philosophies may be considered:

a) the remote server dictates the permissions; the proxy simply passes back what it gets from the remote server.

b) the remote server unveils "everything"; the proxy is responsible for protecting data from unauthorized access.

Of course the latter sounds unreasonable, but it is not. It is possible to imagine scenarios in which a remote host discloses data that can be considered "public" inside an intranet, and a proxy that connects it to the internet may impose additional constraints. To this purpose, the proxy should be able to comply with all the ACL matching criteria that the server supports. This has been achieved with regard to all the criteria supported by slapd except a special subtle case (please drop me a note if you can find other exceptions: <ando@openldap.org>). The rule

```
access to dn="<dn>" attrs=<attr>
    by dnattr=<dnattr> read
    by * none
```

cannot be matched iff the attribute that is being requested, <attr>, is NOT <dnattr>, and the attribute that determines membership, <dnattr>, has not been requested (e.g. in a search)

In fact this ACL is resolved by slapd using the portion of entry it retrieved from the remote server without requiring any further intervention of the backend, so, if the <dnattr> attribute has not been fetched, the match cannot be assessed because the attribute is not present, not because no value matches the requirement!

Note on ACLs and attribute mapping: ACLs are applied to the mapped attributes; for instance, if the attribute locally known as "foo" is mapped to "bar" on a remote server, then local ACLs apply to attribute "foo" and are totally unaware of its remote name. The remote server will check permissions for "bar", and the local server will possibly enforce additional restrictions to "foo".

REWRITING

A string is rewritten according to a set of rules, called a ‘rewrite context’. The rules are based on POSIX (’’extended’’) regular expressions (regex) with substring matching; basic variable substitution and map resolution of substrings is allowed by specific mechanisms detailed in the following. The behavior of pattern matching/substitution can be altered by a set of flags.

The underlying concept is to build a lightweight rewrite module for the slapd server (initially dedicated to the LDAP backend).

Passes

An incoming string is matched against a set of rules. Rules are made of a regex match pattern, a substitution pattern and a set of actions, described by a set of flags. In case of match a string rewriting is performed according to the substitution pattern that allows to refer to substrings matched in the incoming string. The actions, if any, are finally performed. The substitution pattern allows map resolution of substrings. A map is a generic object that maps a substitution pattern to a value. The flags are divided in "Pattern matching Flags" and "Action Flags"; the former alter the regex match pattern behavior while the latter alter the action that is taken after substitution.

Pattern Matching Flags

‘C’ honors case in matching (default is case insensitive)

'R' use POSIX ''basic'' regular expressions (default is ''extended'')

'M{n}' allow no more than **n** recursive passes for a specific rule; does not alter the max total count of passes, so it can only enforce a stricter limit for a specific rule.

Action Flags

':' apply the rule once only (default is recursive)

'@' stop applying rules in case of match; the current rule is still applied recursively; combine with ':' to apply the current rule only once and then stop.

'#' stop current operation if the rule matches, and issue an 'unwilling to perform' error.

'G{n}' jump **n** rules back and forth (watch for loops!). Note that 'G{1}' is implicit in every rule.

'I' ignores errors in rule; this means, in case of error, e.g. issued by a map, the error is treated as a missed match. The 'unwilling to perform' is not overridden.

'U{n}' uses **n** as return code if the rule matches; the flag does not alter the recursive behavior of the rule, so, to have it performed only once, it must be used in combination with ':', e.g. **':U{16}'** returns the value '16' after exactly one execution of the rule, if the pattern matches. As a consequence, its behavior is equivalent to '@', with the return code set to **n**; or, in other words, '@' is equivalent to 'U{0}'. By convention, the freely available codes are above 16 included; the others are reserved.

The ordering of the flags can be significant. For instance: 'IG{2}' means ignore errors and jump two lines ahead both in case of match and in case of error, while 'G{2}I' means ignore errors, but jump two lines ahead only in case of match.

More flags (mainly Action Flags) will be added as needed.

Pattern matching:

See **regex**(7) and/or **re_format**(7).

Substitution Pattern Syntax:

Everything starting with '%' requires substitution;

the only obvious exception is '%%', which is left as is;

the basic substitution is '%d', where 'd' is a digit; 0 means the whole string, while 1-9 is a submatch;

a '%' followed by a '{' invokes an advanced substitution. The pattern is:

'%' '{' [<op>] <name> '(' <substitution> ')' '}'

where <name> must be a legal name for the map, i.e.

<name> ::= [a-z][a-z0-9]* (case insensitive)
<op> ::= '>' '|' '&' '&&' '*' '**' '$'

and <substitution> must be a legal substitution pattern, with no limits on the nesting level.

The operators are:

> sub context invocation; <name> must be a legal, already defined rewrite context name

| external command invocation; <name> must refer to a legal, already defined command name (NOT IMPL.)

& variable assignment; <name> defines a variable in the running operation structure which can be dereferenced later; operator **&** assigns a variable in the rewrite context scope; operator **&&** assigns a variable that scopes the entire session, e.g. its value can be dereferenced later by other rewrite contexts

***** variable dereferencing; <name> must refer to a variable that is defined and assigned for the running operation; operator ***** dereferences a variable scoping the rewrite context; operator ****** dereferences a variable scoping the whole session, e.g. the value is passed across rewrite contexts

$ parameter dereferencing; <name> must refer to an existing parameter; the idea is to make some run-time parameters set by the system available to the rewrite engine, as the client host name, the

bind DN if any, constant parameters initialized at config time, and so on; no parameter is currently set by either **back−ldap** or **back−meta**, but constant parameters can be defined in the configuration file by using the **rewriteParam** directive.

Substitution escaping has been delegated to the '%' symbol, which is used instead of '\' in string substitution patterns because '\' is already escaped by slapd's low level parsing routines; as a consequence, regex escaping requires two '\' symbols, e.g. '**.*\.foo\.bar**' must be written as '**.*\\.foo\\.bar**'.

Rewrite context:

A rewrite context is a set of rules which are applied in sequence. The basic idea is to have an application initialize a rewrite engine (think of Apache's mod_rewrite ...) with a set of rewrite contexts; when string rewriting is required, one invokes the appropriate rewrite context with the input string and obtains the newly rewritten one if no errors occur.

Each basic server operation is associated to a rewrite context; they are divided in two main groups: client −> server and server −> client rewriting.

client -> server:

(default)	if defined and no specific context is available
bindDN	bind
searchBase	search
searchFilter	search
searchFilterAttrDN	search
compareDN	compare
compareAttrDN	compare AVA
addDN	add
addAttrDN	add AVA
modifyDN	modify
modifyAttrDN	modify AVA
modrDN	modrdn
newSuperiorDN	modrdn
deleteDN	delete
exopPasswdDN	password modify extended operation DN if proxy

server -> client:

searchResult	search (only if defined; no default; acts on DN and DN-syntax attributes of search results)
searchAttrDN	search AVA
matchedDN	all ops (only if applicable)

Basic configuration syntax

rewriteEngine { on | off }

If 'on', the requested rewriting is performed; if 'off', no rewriting takes place (an easy way to stop rewriting without altering too much the configuration file).

rewriteContext <context name> [alias <aliased context name>]

<Context name> is the name that identifies the context, i.e. the name used by the application to refer to the set of rules it contains. It is used also to reference sub contexts in string rewriting. A context may alias another one. In this case the alias context contains no rule, and any reference to it will result in accessing the aliased one.

rewriteRule <regex match pattern> <substitution pattern> [<flags>]

Determines how a string can be rewritten if a pattern is matched. Examples are reported below.

Additional configuration syntax:

rewriteMap <map type> <map name> [<map attrs>]
Allows to define a map that transforms substring rewriting into something else. The map is referenced inside the substitution pattern of a rule.

rewriteParam <param name> <param value>
Sets a value with global scope, that can be dereferenced by the command '%{$paramName}'.

rewriteMaxPasses <number of passes> [<number of passes per rule>]
Sets the maximum number of total rewriting passes that can be performed in a single rewrite operation (to avoid loops). A safe default is set to 100; note that reaching this limit is still treated as a success; recursive invocation of rules is simply interrupted. The count applies to the rewriting operation as a whole, not to any single rule; an optional per-rule limit can be set. This limit is overridden by setting specific per-rule limits with the 'M{n}' flag.

Configuration examples:

```
# set to 'off' to disable rewriting
rewriteEngine on

# the rules the "suffixmassage" directive implies
rewriteEngine on
# all dataflow from client to server referring to DNs
rewriteContext default
rewriteRule "(.*)<virtualnamingcontext>$" "%1<realnamingcontext>" ":"
# empty filter rule
rewriteContext searchFilter
# all dataflow from server to client
rewriteContext searchResult
rewriteRule "(.*)<realnamingcontext>$" "%1<virtualnamingcontext>" ":"
rewriteContext searchAttrDN alias searchResult
rewriteContext matchedDN alias searchResult

# Everything defined here goes into the 'default' context.
# This rule changes the naming context of anything sent
# to 'dc=home,dc=net' to 'dc=OpenLDAP, dc=org'

rewriteRule "(.*)dc=home,[ ]?dc=net"
        "%1dc=OpenLDAP, dc=org"  ":"

# since a pretty/normalized DN does not include spaces
# after rdn separators, e.g. ',', this rule suffices:

rewriteRule "(.*)dc=home,dc=net"
        "%1dc=OpenLDAP,dc=org"  ":"

# Start a new context (ends input of the previous one).
# This rule adds blanks between DN parts if not present.
rewriteContext  addBlanks
rewriteRule     "(.*),([^ ].*)" "%1, %2"

# This one eats blanks
rewriteContext  eatBlanks
rewriteRule     "(.*),[ ](.*)" "%1,%2"

# Here control goes back to the default rewrite
# context; rules are appended to the existing ones.
# anything that gets here is piped into rule 'addBlanks'
```

```
rewriteContext  default
rewriteRule     ".*" "%{>addBlanks(%0)}" ":"

# Rewrite the search base according to ‘default’ rules.
rewriteContext  searchBase alias default

# Search results with OpenLDAP DN are rewritten back with
# ‘dc=home,dc=net’ naming context, with spaces eaten.
rewriteContext  searchResult
rewriteRule     "(.*[^ ]?)[ ]?dc=OpenLDAP,[ ]?dc=org"
                "%{>eatBlanks(%1)}dc=home,dc=net"    ":"

# Bind with email instead of full DN: we first need
# an ldap map that turns attributes into a DN (the
# argument used when invoking the map is appended to
# the URI and acts as the filter portion)
rewriteMap ldap attr2dn "ldap://host/dc=my,dc=org?dn?sub"

# Then we need to detect DN made up of a single email,
# e.g. ‘mail=someone@example.com’; note that the rule
# in case of match stops rewriting; in case of error,
# it is ignored.  In case we are mapping virtual
# to real naming contexts, we also need to rewrite
# regular DNs, because the definition of a bindDn
# rewrite context overrides the default definition.
rewriteContext bindDN
rewriteRule "^mail=[^,]+@[^,]+$" "%{attr2dn(%0)}" ":@I"

# This is a rather sophisticated example. It massages a
# search filter in case who performs the search has
# administrative privileges.  First we need to keep
# track of the bind DN of the incoming request, which is
# stored in a variable called ‘binddn’ with session scope,
# and left in place to allow regular binding:
rewriteContext  bindDN
rewriteRule     ".+" "%{&&binddn(%0)}%0" ":"

# A search filter containing ‘uid=’ is rewritten only
# if an appropriate DN is bound.
# To do this, in the first rule the bound DN is
# dereferenced, while the filter is decomposed in a
# prefix, in the value of the ‘uid=<arg>’ AVA, and
# in a suffix. A tag ‘<>’ is appended to the DN.
# If the DN refers to an entry in the ‘ou=admin’ subtree,
# the filter is rewritten OR-ing the ‘uid=<arg>’ with
# ‘cn=<arg>’; otherwise it is left as is. This could be
# useful, for instance, to allow apache’s auth_ldap-1.4
# module to authenticate users with both ‘uid’ and
# ‘cn’, but only if the request comes from a possible
# ‘cn=Web auth,ou=admin,dc=home,dc=net’ user.
rewriteContext searchFilter
rewriteRule "(.*\\()uid=([a-z0-9_]+)(\\).*)"
  "%{**binddn}<>%{&prefix(%1)}%{&arg(%2)}%{&suffix(%3)}"
  ":I"
```

```
rewriteRule "[^,]+,ou=admin,dc=home,dc=net"
  "%{*prefix}|(uid=%{*arg})(cn=%{*arg})%{*suffix}" ":@I"
rewriteRule ".*<>" "%{*prefix}uid=%{*arg}%{*suffix}" ":"

# This example shows how to strip unwanted DN-valued
# attribute values from a search result; the first rule
# matches DN values below "ou=People,dc=example,dc=com";
# in case of match the rewriting exits successfully.
# The second rule matches everything else and causes
# the value to be rejected.
rewriteContext searchResult
rewriteRule ".*,ou=People,dc=example,dc=com" "%0" ":@"
rewriteRule ".*" "" "#"
```

LDAP Proxy resolution (a possible evolution of slapd−ldap(5)):

In case the rewritten DN is an LDAP URI, the operation is initiated towards the host[:port] indicated in the uri, if it does not refer to the local server. E.g.:

```
rewriteRule '^cn=root,.*' '%0'                  'G{3}'
rewriteRule '^cn=[a-l].*' 'ldap://ldap1.my.org/%0' ':@'
rewriteRule '^cn=[m-z].*' 'ldap://ldap2.my.org/%0' ':@'
rewriteRule '.*'          'ldap://ldap3.my.org/%0' ':@'
```

(Rule 1 is simply there to illustrate the 'G{n}' action; it could have been written:

```
rewriteRule '^cn=root,.*' 'ldap://ldap3.my.org/%0' ':@'
```

with the advantage of saving one rewrite pass ...)

ACCESS CONTROL

The **meta** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access checking is delegated to the remote server(s). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

PROXY CACHE OVERLAY

The proxy cache overlay allows caching of LDAP search requests (queries) in a local database. See **slapo-pcache**(5) for details.

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapo−pcache**(5), **slapd**(8), **regex**(7), **re_format**(7).

AUTHOR

Pierangelo Masarati, based on back-ldap by Howard Chu

NAME

slapd-monitor − Monitor backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **monitor** backend to **slapd**(8) is not an actual database; if enabled, it is automatically generated and dynamically maintained by **slapd** with information about the running status of the daemon.

To inspect all monitor information, issue a subtree search with base cn=Monitor, requesting that attributes "+" and "*" are returned. The monitor backend produces mostly operational attributes, and LDAP only returns operational attributes that are explicitly requested. Requesting attribute "+" is an extension which requests all operational attributes.

CONFIGURATION

These **slapd.conf** options apply to the **monitor** backend database. That is, they must follow a "database monitor" line and come before any subsequent "backend" or "database" lines.

As opposed to most databases, the **monitor** database can be instantiated only once, i.e. only one occurrence of "database monitor" can occur in the **slapd.conf**(5) file. Moreover, the suffix of the database cannot be explicitly set by means of the **suffix** directive. The suffix is automatically set to "*cn=Monitor*".

The **monitor** database honors the **rootdn** and the **rootpw** directives, and the usual ACL directives, e.g. the **access** directive.

Other database options are described in the **slapd.conf**(5) manual page.

USAGE

The usage is:

1) enable the **monitor** backend at configure:

```
configure --enable-monitor
```

2) activate the **monitor** database in the **slapd.conf**(5) file:

```
database monitor
```

3) add ACLs as detailed in **slapd.access**(5) to control access to the database, e.g.:

```
access to dn.subtree="cn=Monitor"
        by dn.exact="uid=Admin,dc=my,dc=org" write
        by users read
        by * none
```

4) ensure that the **core.schema** file is loaded.

The **monitor** backend relies on some standard track attributeTypes that must be already defined when the backend is started.

ACCESS CONTROL

The **monitor** backend honors access control semantics as indicated in **slapd.access**(5), including the **disclose** access privilege, on all currently implemented operations.

KNOWN LIMITATIONS

The **monitor** backend does not honor size/time limits in search operations.

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd.access**(5), **slapd**(8), **ldap**(3).

ACKNOWLEDGEMENTS

NAME

slapd-monitor − Monitor backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **monitor** backend to **slapd**(8) is not an actual database; if enabled, it is automatically generated and dynamically maintained by **slapd** with information about the running status of the daemon.

To inspect all monitor information, issue a subtree search with base cn=Monitor, requesting that attributes "+" and "*" are returned. The monitor backend produces mostly operational attributes, and LDAP only returns operational attributes that are explicitly requested. Requesting attribute "+" is an extension which requests all operational attributes.

CONFIGURATION

These **slapd.conf** options apply to the **monitor** backend database. That is, they must follow a "database monitor" line and come before any subsequent "backend" or "database" lines.

As opposed to most databases, the **monitor** database can be instantiated only once, i.e. only one occurrence of "database monitor" can occur in the **slapd.conf**(5) file. Moreover, the suffix of the database cannot be explicitly set by means of the **suffix** directive. The suffix is automatically set to "*cn=Monitor*".

The **monitor** database honors the **rootdn** and the **rootpw** directives, and the usual ACL directives, e.g. the **access** directive.

Other database options are described in the **slapd.conf**(5) manual page.

USAGE

The usage is:

1) enable the **monitor** backend at configure:

```
configure --enable-monitor
```

2) activate the **monitor** database in the **slapd.conf**(5) file:

```
database monitor
```

3) add ACLs as detailed in **slapd.access**(5) to control access to the database, e.g.:

```
access to dn.subtree="cn=Monitor"
        by dn.exact="uid=Admin,dc=my,dc=org" write
        by users read
        by * none
```

4) ensure that the **core.schema** file is loaded.

The **monitor** backend relies on some standard track attributeTypes that must be already defined when the backend is started.

ACCESS CONTROL

The **monitor** backend honors access control semantics as indicated in **slapd.access**(5), including the **disclose** access privilege, on all currently implemented operations.

KNOWN LIMITATIONS

The **monitor** backend does not honor size/time limits in search operations.

FILES

/etc/openldap/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd.access**(5), **slapd**(8), **ldap**(3).

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapd-null − Null backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Null backend to **slapd**(8) is surely the most useful part of **slapd**:
- Searches return success but no entries.
- Compares return compareFalse.
- Updates return success (unless readonly is on) but do nothing.
- Binds other than as the rootdn fail unless the database option "bind on" is given.
- The **slapadd**(8) and **slapcat**(8) tools are equally exciting.

Inspired by the /dev/null device.

CONFIGURATION

This **slapd.conf** option applies to the NULL backend database. That is, it must follow a "database null" line and come before any subsequent "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

bind <on/off>

Allow binds as any DN in this backend’s suffix, with any password. The default is "off".

EXAMPLE

Here is a possible slapd.conf extract using the Null backend:

```
database null
suffix   "cn=Nothing"
bind     on
```

ACCESS CONTROL

The **null** backend does not honor any of the access control semantics described in **slapd.access**(5).

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **slapadd**(8), **slapcat**(8).

NAME

slapd-null − Null backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Null backend to **slapd**(8) is surely the most useful part of **slapd**:
- Searches return success but no entries.
- Compares return compareFalse.
- Updates return success (unless readonly is on) but do nothing.
- Binds other than as the rootdn fail unless the database option "bind on" is given.
- The **slapadd**(8) and **slapcat**(8) tools are equally exciting.

Inspired by the /dev/null device.

CONFIGURATION

This **slapd.conf** option applies to the NULL backend database. That is, it must follow a "database null" line and come before any subsequent "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

bind <on/off>

Allow binds as any DN in this backend's suffix, with any password. The default is "off".

EXAMPLE

Here is a possible slapd.conf extract using the Null backend:

```
database null
suffix   "cn=Nothing"
bind     on
```

ACCESS CONTROL

The **null** backend does not honor any of the access control semantics described in **slapd.access**(5).

FILES

/etc/openldap/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **slapadd**(8), **slapcat**(8).

NAME

slapd-passwd − /etc/passwd backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The PASSWD backend to **slapd**(8) serves up the user account information listed in the system **passwd**(5) file. This backend is provided for demonstration purposes only. The DN of each entry is "uid=<username>,<suffix>". Note that non-base searches scan the the entire passwd file, and are best suited for hosts with small passwd files.

CONFIGURATION

This **slapd.conf** option applies to the PASSWD backend database. That is, it must follow a "database passwd" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

file <filename>

Specifies an alternate passwd file to use. The default is **/etc/passwd**.

ACCESS CONTROL

The **passwd** backend does not honor any of the access control semantics described in **slapd.access**(5). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

FILES

ETCDIR/slapd.conf

default slapd configuration file

/etc/passwd

user account information

SEE ALSO

slapd.conf(5), **slapd**(8), **passwd**(5).

NAME

slapd-passwd − /etc/passwd backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The PASSWD backend to **slapd**(8) serves up the user account information listed in the system **passwd**(5) file. This backend is provided for demonstration purposes only. The DN of each entry is "uid=<username>,<suffix>". Note that non-base searches scan the the entire passwd file, and are best suited for hosts with small passwd files.

CONFIGURATION

This **slapd.conf** option applies to the PASSWD backend database. That is, it must follow a "database passwd" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

file <filename>

Specifies an alternate passwd file to use. The default is **/etc/passwd**.

ACCESS CONTROL

The **passwd** backend does not honor any of the access control semantics described in **slapd.access**(5). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

FILES

/etc/openldap/slapd.conf

default slapd configuration file

/etc/passwd

user account information

SEE ALSO

slapd.conf(5), **slapd**(8), **passwd**(5).

NAME

slapd-perl − Perl backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Perl backend to **slapd**(8) works by embedding a **perl**(1) interpreter into **slapd**(8). Any perl database section of the configuration file **slapd.conf**(5) must then specify what Perl module to use. **Slapd** then creates a new Perl object that handles all the requests for that particular instance of the backend.

You will need to create a method for each one of the following actions:

```
* new       # creates a new object,
* search    # performs the ldap search,
* compare   # does a compare,
* modify    # modifies an entry,
* add       # adds an entry to backend,
* modrdn    # modifies an entry's rdn,
* delete    # deletes an ldap entry,
* config    # process unknown config file lines,
* init      # called after backend is initialized.
```

Unless otherwise specified, the methods return the result code which will be returned to the client. Unimplemented actions can just return unwillingToPerform (53).

new This method is called when the configuration file encounters a **perlmod** line. The module in that line is then effectively 'use'd into the perl interpreter, then the **new** method is called to create a new object. Note that multiple instances of that object may be instantiated, as with any perl object. The **new** method receives the class name as argument.

search This method is called when a search request comes from a client. It arguments are as follows:

* object reference
* base DN
* scope
* alias dereferencing policy
* size limit
* time limit
* filter string
* attributes only flag (1 for yes)
* list of attributes to return (may be empty)

Return value: (resultcode, ldif-entry, ldif-entry, ...)

compare

This method is called when a compare request comes from a client. Its arguments are as follows.

* object reference
* dn
* attribute assertion string

modify This method is called when a modify request comes from a client. Its arguments are as follows.

* object reference
* dn
* a list formatted as follows
  ({ "ADD" | "DELETE" | "REPLACE" },
  attributetype, value...)...

add This method is called when a add request comes from a client. Its arguments are as follows.

* object reference
* entry in string format

modrdn
This method is called when a modrdn request comes from a client. Its arguments are as follows.
* object reference
* dn
* new rdn
* delete old dn flag (1 means yes)

delete This method is called when a delete request comes from a client. Its arguments are as follows.
* object reference
* dn

config This method is called with unknown **slapd.conf**(5) configuration file lines. Its arguments are as follows.
* object reference
* array of arguments on line

Return value: nonzero if this is not a valid option.

init This method is called after backend is initialized. Its argument is as follows.
* object reference

Return value: nonzero if initialization failed.

CONFIGURATION

These **slapd.conf** options apply to the PERL backend database. That is, they must follow a "database perl" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

perlModulePath /path/to/libs
Add the path to the @INC variable.

perlModule ModName
'Use' the module name ModName from ModName.pm

filterSearchResults
Search results are candidates that need to be filtered (with the filter in the search request), rather than search results to be returned directly to the client.

EXAMPLE

There is an example Perl module 'SampleLDAP' in the slapd/back-perl/ directory in the OpenLDAP source tree.

ACCESS CONTROL

The **passwd** backend does not honor any of the access control semantics described in **slapd.access**(5); all access control is delegated to the underlying PERL scripting. Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

WARNING

The interface of this backend to the perl module MAY change. Any suggestions would greatly be appreciated.

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **perl**(1).

NAME

slapd-perl − Perl backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Perl backend to **slapd**(8) works by embedding a **perl**(1) interpreter into **slapd**(8). Any perl database section of the configuration file **slapd.conf**(5) must then specify what Perl module to use. **Slapd** then creates a new Perl object that handles all the requests for that particular instance of the backend.

You will need to create a method for each one of the following actions:

```
* new       # creates a new object,
* search    # performs the ldap search,
* compare   # does a compare,
* modify    # modifies an entry,
* add       # adds an entry to backend,
* modrdn    # modifies an entry's rdn,
* delete    # deletes an ldap entry,
* config    # process unknown config file lines,
* init      # called after backend is initialized.
```

Unless otherwise specified, the methods return the result code which will be returned to the client. Unimplemented actions can just return unwillingToPerform (53).

new This method is called when the configuration file encounters a **perlmod** line. The module in that line is then effectively 'use'd into the perl interpreter, then the **new** method is called to create a new object. Note that multiple instances of that object may be instantiated, as with any perl object. The **new** method receives the class name as argument.

search This method is called when a search request comes from a client. It arguments are as follows:

* object reference
* base DN
* scope
* alias dereferencing policy
* size limit
* time limit
* filter string
* attributes only flag (1 for yes)
* list of attributes to return (may be empty)

Return value: (resultcode, ldif-entry, ldif-entry, ...)

compare

This method is called when a compare request comes from a client. Its arguments are as follows.

* object reference
* dn
* attribute assertion string

modify This method is called when a modify request comes from a client. Its arguments are as follows.

* object reference
* dn
* a list formatted as follows

  ({ "ADD" | "DELETE" | "REPLACE" },
  attributetype, value...)...

add This method is called when a add request comes from a client. Its arguments are as follows.

* object reference
* entry in string format

modrdn
This method is called when a modrdn request comes from a client. Its arguments are as follows.
* object reference
* dn
* new rdn
* delete old dn flag (1 means yes)

delete This method is called when a delete request comes from a client. Its arguments are as follows.
* object reference
* dn

config This method is called with unknown **slapd.conf**(5) configuration file lines. Its arguments are as follows.
* object reference
* array of arguments on line

Return value: nonzero if this is not a valid option.

init This method is called after backend is initialized. Its argument is as follows.
* object reference

Return value: nonzero if initialization failed.

CONFIGURATION

These **slapd.conf** options apply to the PERL backend database. That is, they must follow a "database perl" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

perlModulePath /path/to/libs
Add the path to the @INC variable.

perlModule ModName
'Use' the module name ModName from ModName.pm

filterSearchResults
Search results are candidates that need to be filtered (with the filter in the search request), rather than search results to be returned directly to the client.

EXAMPLE

There is an example Perl module 'SampleLDAP' in the slapd/back-perl/ directory in the OpenLDAP source tree.

ACCESS CONTROL

The **passwd** backend does not honor any of the access control semantics described in **slapd.access**(5); all access control is delegated to the underlying PERL scripting. Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

WARNING

The interface of this backend to the perl module MAY change. Any suggestions would greatly be appreciated.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **perl**(1).

NAME

slapd-relay − relay backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The primary purpose of this **slapd**(8) backend is to map a naming context defined in a database running in the same **slapd**(8) instance into a virtual naming context, with attributeType and objectClass manipulation, if required. It requires the **slapo-rwm**(5) overlay.

This backend and the above mentioned overlay are experimental.

CONFIGURATION

The following **slapd.conf** directives apply to the relay backend database. That is, they must follow a "database relay" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page; only the **suffix** directive is allowed by the *relay* backend.

relay <real naming context>

The naming context of the database that is presented under a virtual naming context. The presence of this directive implies that one specific database, i.e. the one serving the **real naming context**, will be presented under a virtual naming context.

MASSAGING

The **relay** database does not automatically rewrite the naming context of requests and responses. For this purpose, the **slapo-rwm**(5) overlay must be explicitly instantiated, and configured as appropriate. Usually, the **rwm-suffixmassage** directive suffices if only naming context rewriting is required.

ACCESS RULES

One important issue is that access rules are based on the identity that issued the operation. After massaging from the virtual to the real naming context, the frontend sees the operation as performed by the identity in the real naming context. Moreover, since **back-relay** bypasses the real database frontend operations by short-circuiting operations through the internal backend API, the original database access rules do not apply but in selected cases, i.e. when the backend itself applies access control. As a consequence, the instances of the relay database must provide own access rules that are consistent with those of the original database, possibly adding further specific restrictions. So, access rules in the **relay** database must refer to identities in the real naming context. Examples are reported in the EXAMPLES section.

SCENARIOS

If no **relay** directive is given, the *relay* database does not refer to any specific database, but the most appropriate one is looked-up after rewriting the request DN for the operation that is being handled.

This allows to write carefully crafted rewrite rules that cause some of the requests to be directed to one database, and some to another; e.g., authentication can be mapped to one database, and searches to another, or different target databases can be selected based on the DN of the request, and so.

Another possibility is to map the same operation to different databases based on details of the virtual naming context, e.g. groups on one database and persons on another.

EXAMPLES

To implement a plain virtual naming context mapping that refers to a single database, use

```
database        relay
suffix          "dc=virtual,dc=naming,dc=context"
relay           "dc=real,dc=naming,dc=context"
overlay         rwm
rwm-suffixmassage       "dc=real,dc=naming,dc=context"
```

To implement a plain virtual naming context mapping that looks up the real naming context for each

operation, use

```
database        relay
suffix          "dc=virtual,dc=naming,dc=context"
overlay         rwm
rwm-suffixmassage   "dc=real,dc=naming,dc=context"
```

This is useful, for instance, to relay different databases that share the terminal portion of the naming context (the one that is rewritten).

To implement the old-fashioned suffixalias, e.g. mapping the virtual to the real naming context, but not the results back from the real to the virtual naming context, use

```
database        relay
suffix          "dc=virtual,dc=naming,dc=context"
relay           "dc=real,dc=naming,dc=context"
overlay         rwm
rwm-rewriteEngine   on
rwm-rewriteContext  default
rwm-rewriteRule     "dc=virtual,dc=naming,dc=context"
                "dc=real,dc=naming,dc=context" ":@"
rwm-rewriteContext  searchFilter
rwm-rewriteContext  searchEntryDN
rwm-rewriteContext  searchAttrDN
rwm-rewriteContext  matchedDN
```

Note that the **slapo-rwm**(5) overlay is instantiated, but the rewrite rules are written explicitly, rather than automatically as with the **rwm-suffixmassage** statement, to map all the virtual to real naming context data flow, but none of the real to virtual.

Access rules:

```
database        bdb
suffix          "dc=example,dc=com"
# skip...
access to dn.subtree="dc=example,dc=com"
    by dn.exact="cn=Supervisor,dc=example,dc=com" write
    by * read

database        relay
suffix          "o=Example,c=US"
relay           "dc=example,dc=com"
overlay         rwm
rwm-suffixmassage   "dc=example,dc=com"
# skip ...
access to dn.subtree="o=Example,c=US"
    by dn.exact="cn=Supervisor,dc=example,dc=com" write
    by dn.exact="cn=Relay Supervisor,dc=example,dc=com" write
    by * read
```

Note that, in both databases, the identities (the **<who>** clause) are in the **real naming context**, i.e. **'dc=example,dc=com'**, while the targets (the **<what>** clause) are in the **real** and in the **virtual naming context**, respectively.

ACCESS CONTROL

The **relay** backend does not honor any of the access control semantics described in **slapd.access**(5); all access control is delegated to the relayed database(s). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapo-rwm**(5), **slapd**(8).

NAME

slapd-relay − relay backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The primary purpose of this **slapd**(8) backend is to map a naming context defined in a database running in the same **slapd**(8) instance into a virtual naming context, with attributeType and objectClass manipulation, if required. It requires the **slapo-rwm**(5) overlay.

This backend and the above mentioned overlay are experimental.

CONFIGURATION

The following **slapd.conf** directives apply to the relay backend database. That is, they must follow a "database relay" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page; only the **suffix** directive is allowed by the *relay* backend.

relay <real naming context>

The naming context of the database that is presented under a virtual naming context. The presence of this directive implies that one specific database, i.e. the one serving the **real naming context**, will be presented under a virtual naming context.

MASSAGING

The **relay** database does not automatically rewrite the naming context of requests and responses. For this purpose, the **slapo-rwm**(5) overlay must be explicitly instantiated, and configured as appropriate. Usually, the **rwm-suffixmassage** directive suffices if only naming context rewriting is required.

ACCESS RULES

One important issue is that access rules are based on the identity that issued the operation. After massaging from the virtual to the real naming context, the frontend sees the operation as performed by the identity in the real naming context. Moreover, since **back-relay** bypasses the real database frontend operations by short-circuiting operations through the internal backend API, the original database access rules do not apply but in selected cases, i.e. when the backend itself applies access control. As a consequence, the instances of the relay database must provide own access rules that are consistent with those of the original database, possibly adding further specific restrictions. So, access rules in the **relay** database must refer to identities in the real naming context. Examples are reported in the EXAMPLES section.

SCENARIOS

If no **relay** directive is given, the *relay* database does not refer to any specific database, but the most appropriate one is looked-up after rewriting the request DN for the operation that is being handled.

This allows to write carefully crafted rewrite rules that cause some of the requests to be directed to one database, and some to another; e.g., authentication can be mapped to one database, and searches to another, or different target databases can be selected based on the DN of the request, and so.

Another possibility is to map the same operation to different databases based on details of the virtual naming context, e.g. groups on one database and persons on another.

EXAMPLES

To implement a plain virtual naming context mapping that refers to a single database, use

```
database        relay
suffix          "dc=virtual,dc=naming,dc=context"
relay           "dc=real,dc=naming,dc=context"
overlay         rwm
rwm-suffixmassage       "dc=real,dc=naming,dc=context"
```

To implement a plain virtual naming context mapping that looks up the real naming context for each

operation, use

```
database        relay
suffix          "dc=virtual,dc=naming,dc=context"
overlay         rwm
rwm-suffixmassage   "dc=real,dc=naming,dc=context"
```

This is useful, for instance, to relay different databases that share the terminal portion of the naming context (the one that is rewritten).

To implement the old-fashioned suffixalias, e.g. mapping the virtual to the real naming context, but not the results back from the real to the virtual naming context, use

```
database        relay
suffix          "dc=virtual,dc=naming,dc=context"
relay           "dc=real,dc=naming,dc=context"
overlay         rwm
rwm-rewriteEngine   on
rwm-rewriteContext  default
rwm-rewriteRule     "dc=virtual,dc=naming,dc=context"
                "dc=real,dc=naming,dc=context" ":@"
rwm-rewriteContext  searchFilter
rwm-rewriteContext  searchEntryDN
rwm-rewriteContext  searchAttrDN
rwm-rewriteContext  matchedDN
```

Note that the **slapo-rwm**(5) overlay is instantiated, but the rewrite rules are written explicitly, rather than automatically as with the **rwm-suffixmassage** statement, to map all the virtual to real naming context data flow, but none of the real to virtual.

Access rules:

```
database        bdb
suffix          "dc=example,dc=com"
# skip...
access to dn.subtree="dc=example,dc=com"
    by dn.exact="cn=Supervisor,dc=example,dc=com" write
    by * read

database        relay
suffix          "o=Example,c=US"
relay           "dc=example,dc=com"
overlay         rwm
rwm-suffixmassage   "dc=example,dc=com"
# skip ...
access to dn.subtree="o=Example,c=US"
    by dn.exact="cn=Supervisor,dc=example,dc=com" write
    by dn.exact="cn=Relay Supervisor,dc=example,dc=com" write
    by * read
```

Note that, in both databases, the identities (the **<who>** clause) are in the **real naming context**, i.e. **'dc=example,dc=com'**, while the targets (the **<what>** clause) are in the **real** and in the **virtual naming context**, respectively.

ACCESS CONTROL

The **relay** backend does not honor any of the access control semantics described in **slapd.access**(5); all access control is delegated to the relayed database(s). Only **read (=r)** access to the **entry** pseudo-attribute and to the other attribute values of the entries returned by the **search** operation is honored, which is performed by the frontend.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapo-rwm**(5), **slapd**(8).

NAME

slapd-shell − Shell backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Shell backend to **slapd**(8) executes external programs to implement operations, and is designed to make it easy to tie an existing database to the **slapd** front-end.

This backend is primarily intended to be used in prototypes.

WARNING

The **abandon** shell command has been removed since OpenLDAP 2.1.

CONFIGURATION

These **slapd.conf** options apply to the SHELL backend database. That is, they must follow a "database shell" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

These options specify the pathname and arguments of the program to execute in response to the given LDAP operation. Each option is followed by the input lines that the program receives:

add <pathname> <argument>...

ADD
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
<entry in LDIF format>

bind <pathname> <argument>...

BIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
method: <method number>
credlen: <length of <credentials>>
cred: <credentials>

compare <pathname> <argument>...

COMPARE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<attribute>: <value>

delete <pathname> <argument>...

DELETE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>

modify <pathname> <argument>...

MODIFY
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<repeat {
<"add"/"delete"/"replace">: <attribute>
<repeat { <attribute>: <value> }>
-
}>

modrdn <pathname> <argument>...
MODRDN
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
newrdn: <new RDN>
deleteoldrdn: <0 or 1>
<if new superior is specified: "newSuperior: <DN>">

search <pathname> <argument>...
SEARCH
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
base: <base DN>
scope: <0-2, see ldap.h>
deref: <0-3, see ldap.h>
sizelimit: <size limit>
timelimit: <time limit>
filter: <filter>
attrsonly: <0 or 1>
attrs: <"all" or space-separated attribute list>

unbind <pathname> <argument>...
UNBIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <bound DN>

Note that you need only supply configuration lines for those commands you want the backend to handle. Operations for which a command is not supplied will be refused with an "unwilling to perform" error.

The **search** command should output the entries in LDIF format, each entry followed by a blank line, and after these the RESULT below.

All commands except **unbind** should then output:
RESULT
code: <integer>
matched: <matched DN>
info: <text>
where only the RESULT line is mandatory. Lines starting with '#' or 'DEBUG:' are ignored.

ACCESS CONTROL

The **shell** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access to objects is checked by using a dummy object that contains only the DN, so access rules that rely on the contents of the object are not honored. In detail:

The **add** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry.

The **bind** operation requires **auth (=x)** access to the **entry** pseudo-attribute of the entry whose identity is being assessed; **auth (=x)** access to the credentials is not checked, but rather delegated to the underlying shell script.

The **compare** operation requires **read (=r)** access (FIXME: wouldn't **compare (=c)** be a more appropriate choice?) to the **entry** pseudo-attribute of the object whose value is being asserted; **compare (=c)** access to the attribute whose value is being asserted is not checked.

The **delete** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry.

The **modify** operation requires **write (=w)** access to the **entry** pseudo-attribute; **write (=w)** access to the specific attributes that are modified is not checked.

The **modrdn** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry, nor to that of the new parent, if different; **write (=w)** access to the distinguished values of the naming attributes is not checked.

The **search** operation does not require **search (=s)** access to the **entry** pseudo_attribute of the searchBase; **search (=s)** access to the attributes and values used in the filter is not checked.

EXAMPLE

There is an example search script in the slapd/back-shell/ directory in the OpenLDAP source tree.

LIMITATIONS

The shell backend does not support threaded environments. When using the shell backend, **slapd**(8) should be built *--without-threads*.

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **sh**(1).

NAME

slapd-shell − Shell backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Shell backend to **slapd**(8) executes external programs to implement operations, and is designed to make it easy to tie an existing database to the **slapd** front-end.

This backend is primarily intended to be used in prototypes.

WARNING

The **abandon** shell command has been removed since OpenLDAP 2.1.

CONFIGURATION

These **slapd.conf** options apply to the SHELL backend database. That is, they must follow a "database shell" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

These options specify the pathname and arguments of the program to execute in response to the given LDAP operation. Each option is followed by the input lines that the program receives:

add <pathname> <argument>...

```
ADD
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
<entry in LDIF format>
```

bind <pathname> <argument>...

```
BIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
method: <method number>
credlen: <length of <credentials>>
cred: <credentials>
```

compare <pathname> <argument>...

```
COMPARE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<attribute>: <value>
```

delete <pathname> <argument>...

```
DELETE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
```

modify <pathname> <argument>...

```
MODIFY
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<repeat {
   <"add"/"delete"/"replace">: <attribute>
   <repeat { <attribute>: <value> }>
   -
}>
```

modrdn <pathname> <argument>...

MODRDN
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
newrdn: <new RDN>
deleteoldrdn: <0 or 1>
<if new superior is specified: "newSuperior: <DN>">

search <pathname> <argument>...

SEARCH
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
base: <base DN>
scope: <0-2, see ldap.h>
deref: <0-3, see ldap.h>
sizelimit: <size limit>
timelimit: <time limit>
filter: <filter>
attrsonly: <0 or 1>
attrs: <"all" or space-separated attribute list>

unbind <pathname> <argument>...

UNBIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <bound DN>

Note that you need only supply configuration lines for those commands you want the backend to handle. Operations for which a command is not supplied will be refused with an "unwilling to perform" error.

The **search** command should output the entries in LDIF format, each entry followed by a blank line, and after these the RESULT below.

All commands except **unbind** should then output:

RESULT
code: <integer>
matched: <matched DN>
info: <text>

where only the RESULT line is mandatory. Lines starting with '#' or 'DEBUG:' are ignored.

ACCESS CONTROL

The **shell** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access to objects is checked by using a dummy object that contains only the DN, so access rules that rely on the contents of the object are not honored. In detail:

The **add** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry.

The **bind** operation requires **auth (=x)** access to the **entry** pseudo-attribute of the entry whose identity is being assessed; **auth (=x)** access to the credentials is not checked, but rather delegated to the underlying shell script.

The **compare** operation requires **read (=r)** access (FIXME: wouldn't **compare (=c)** be a more appropriate choice?) to the **entry** pseudo-attribute of the object whose value is being asserted; **compare (=c)** access to the attribute whose value is being asserted is not checked.

The **delete** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry.

The **modify** operation requires **write (=w)** access to the **entry** pseudo-attribute; **write (=w)** access to the specific attributes that are modified is not checked.

The **modrdn** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry, nor to that of the new parent, if different; **write (=w)** access to the distinguished values of the naming attributes is not checked.

The **search** operation does not require **search (=s)** access to the **entry** pseudo_attribute of the searchBase; **search (=s)** access to the attributes and values used in the filter is not checked.

EXAMPLE

There is an example search script in the slapd/back-shell/ directory in the OpenLDAP source tree.

LIMITATIONS

The shell backend does not support threaded environments. When using the shell backend, **slapd**(8) should be built *--without-threads*.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8), **sh**(1).

NAME

slapd-sock − Socket backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Socket backend to **slapd**(8) uses an external program to handle queries, similarly to **slapd-shell**(5). However, in this case the external program listens on a Unix domain socket. This makes it possible to have a pool of processes, which persist between requests. This allows multithreaded operation and a higher level of efficiency. The external program must have been started independently; **slapd**(8) itself will not start it.

CONFIGURATION

These **slapd.conf** options apply to the SOCK backend database. That is, they must follow a "database sock" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

extensions [binddn | peername | ssf]*

Enables the sending of additional meta-attributes with each request.
binddn: <bound DN>
peername: IP=<address>:<port>
ssf: <SSF value>

socketpath <pathname>

Gives the path to a Unix domain socket to which the commands will be sent and from which replies are received.

PROTOCOL

The protocol is essentially the same as **slapd-shell**(5) with the addition of a newline to terminate the command parameters. The following commands are sent:

ADD
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
<entry in LDIF format>
<blank line>

BIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
method: <method number>
credlen: <length of <credentials>>
cred: <credentials>
<blank line>

COMPARE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<attribute>: <value>
<blank line>

DELETE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<blank line>

MODIFY
msgid: <message id>

<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<repeat {
<"add"/"delete"/"replace">: <attribute>
<repeat { <attribute>: <value> }>
-
}>
<blank line>

MODRDN
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
newrdn: <new RDN>
deleteoldrdn: <0 or 1>
<if new superior is specified: "newSuperior: <DN>">
<blank line>

SEARCH
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
base: <base DN>
scope: <0-2, see ldap.h>
deref: <0-3, see ldap.h>
sizelimit: <size limit>
timelimit: <time limit>
filter: <filter>
attrsonly: <0 or 1>
attrs: <"all" or space-separated attribute list>
<blank line>

UNBIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
<blank line>

The commands - except **unbind** - should output:

RESULT
code: <integer>
matched: <matched DN>
info: <text>

where only RESULT is mandatory, and then close the socket. The **search** RESULT should be preceded by the entries in LDIF format, each entry followed by a blank line. Lines starting with '#' or 'DEBUG:' are ignored.

ACCESS CONTROL

The **sock** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access to objects is checked by using a dummy object that contains only the DN, so access rules that rely on the contents of the object are not honored. In detail:

The **add** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry.

The **bind** operation requires **auth (=x)** access to the **entry** pseudo-attribute of the entry whose identity is being assessed; **auth (=x)** access to the credentials is not checked, but rather delegated to the underlying program.

The **compare** operation requires **compare (=c)** access to the **entry** pseudo-attribute of the object whose value is being asserted; **compare (=c)** access to the attribute whose value is being asserted is not checked.

The **delete** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent

entry.

The **modify** operation requires **write (=w)** access to the **entry** pseudo-attribute; **write (=w)** access to the specific attributes that are modified is not checked.

The **modrdn** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry, nor to that of the new parent, if different; **write (=w)** access to the distinguished values of the naming attributes is not checked.

The **search** operation does not require **search (=s)** access to the **entry** pseudo_attribute of the searchBase; **search (=s)** access to the attributes and values used in the filter is not checked.

EXAMPLE

There is an example script in the slapd/back-sock/ directory in the OpenLDAP source tree.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8).

AUTHOR

Brian Candler

NAME

slapd-sock − Socket backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Socket backend to **slapd**(8) uses an external program to handle queries, similarly to **slapd-shell**(5). However, in this case the external program listens on a Unix domain socket. This makes it possible to have a pool of processes, which persist between requests. This allows multithreaded operation and a higher level of efficiency. The external program must have been started independently; **slapd**(8) itself will not start it.

CONFIGURATION

These **slapd.conf** options apply to the SOCK backend database. That is, they must follow a "database sock" line and come before any subsequent "backend" or "database" lines. Other database options are described in the **slapd.conf**(5) manual page.

extensions [binddn | peername | ssf]*

Enables the sending of additional meta-attributes with each request.
binddn: <bound DN>
peername: IP=<address>:<port>
ssf: <SSF value>

socketpath <pathname>

Gives the path to a Unix domain socket to which the commands will be sent and from which replies are received.

PROTOCOL

The protocol is essentially the same as **slapd-shell**(5) with the addition of a newline to terminate the command parameters. The following commands are sent:

ADD
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
<entry in LDIF format>
<blank line>

BIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
method: <method number>
credlen: <length of <credentials>>
cred: <credentials>
<blank line>

COMPARE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<attribute>: <value>
<blank line>

DELETE
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<blank line>

MODIFY
msgid: <message id>

<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
<repeat {
<"add"/"delete"/"replace">: <attribute>
<repeat { <attribute>: <value> }>
-
}>
<blank line>

MODRDN
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
dn: <DN>
newrdn: <new RDN>
deleteoldrdn: <0 or 1>
<if new superior is specified: "newSuperior: <DN>">
<blank line>

SEARCH
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
base: <base DN>
scope: <0-2, see ldap.h>
deref: <0-3, see ldap.h>
sizelimit: <size limit>
timelimit: <time limit>
filter: <filter>
attrsonly: <0 or 1>
attrs: <"all" or space-separated attribute list>
<blank line>

UNBIND
msgid: <message id>
<repeat { "suffix:" <database suffix DN> }>
<blank line>

The commands - except **unbind** - should output:

RESULT
code: <integer>
matched: <matched DN>
info: <text>

where only RESULT is mandatory, and then close the socket. The **search** RESULT should be preceded by the entries in LDIF format, each entry followed by a blank line. Lines starting with '#' or 'DEBUG:' are ignored.

ACCESS CONTROL

The **sock** backend does not honor all ACL semantics as described in **slapd.access**(5). In general, access to objects is checked by using a dummy object that contains only the DN, so access rules that rely on the contents of the object are not honored. In detail:

The **add** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry.

The **bind** operation requires **auth (=x)** access to the **entry** pseudo-attribute of the entry whose identity is being assessed; **auth (=x)** access to the credentials is not checked, but rather delegated to the underlying program.

The **compare** operation requires **compare (=c)** access to the **entry** pseudo-attribute of the object whose value is being asserted; **compare (=c)** access to the attribute whose value is being asserted is not checked.

The **delete** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent

entry.

The **modify** operation requires **write (=w)** access to the **entry** pseudo-attribute; **write (=w)** access to the specific attributes that are modified is not checked.

The **modrdn** operation does not require **write (=w)** access to the **children** pseudo-attribute of the parent entry, nor to that of the new parent, if different; **write (=w)** access to the distinguished values of the naming attributes is not checked.

The **search** operation does not require **search (=s)** access to the **entry** pseudo_attribute of the searchBase; **search (=s)** access to the attributes and values used in the filter is not checked.

EXAMPLE

There is an example script in the slapd/back-sock/ directory in the OpenLDAP source tree.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8).

AUTHOR

Brian Candler

NAME

slapd-sql − SQL backend to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The primary purpose of this **slapd**(8) backend is to PRESENT information stored in some RDBMS as an LDAP subtree without any programming (some SQL and maybe stored procedures can't be considered programming, anyway ;).

That is, for example, when you (some ISP) have account information you use in an RDBMS, and want to use modern solutions that expect such information in LDAP (to authenticate users, make email lookups etc.). Or you want to synchronize or distribute information between different sites/applications that use RDBMSes and/or LDAP. Or whatever else...

It is NOT designed as a general-purpose backend that uses RDBMS instead of BerkeleyDB (as the standard BDB backend does), though it can be used as such with several limitations. You can take a look at **http://www.openldap.org/faq/index.cgi?file=378** (OpenLDAP FAQ-O-Matic/General LDAP FAQ/Directories vs. conventional databases) to find out more on this point.

The idea (detailed below) is to use some meta-information to translate LDAP queries to SQL queries, leaving relational schema untouched, so that old applications can continue using it without any modifications. This allows SQL and LDAP applications to inter-operate without replication, and exchange data as needed.

The SQL backend is designed to be tunable to virtually any relational schema without having to change source (through that meta-information mentioned). Also, it uses ODBC to connect to RDBMSes, and is highly configurable for SQL dialects RDBMSes may use, so it may be used for integration and distribution of data on different RDBMSes, OSes, hosts etc., in other words, in highly heterogeneous environment.

This backend is *experimental*.

CONFIGURATION

These **slapd.conf** options apply to the SQL backend database, which means that they must follow a "database sql" line and come before any subsequent "backend" or "database" lines. Other database options not specific to this backend are described in the **slapd.conf**(5) manual page.

DATA SOURCE CONFIGURATION

dbname <datasource name>

The name of the ODBC datasource to use.

dbhost <hostname>
dbpasswd <password>
dbuser <username>

The three above options are generally unneeded, because this information is taken from the datasource specified by the **dbname** directive. They allow to override datasource settings. Also, several RDBMS' drivers tend to require explicit passing of user/password, even if those are given in datasource (Note: **dbhost** is currently ignored).

SCOPING CONFIGURATION

These options specify SQL query templates for scoping searches.

subtree_cond <SQL expression>

Specifies a where-clause template used to form a subtree search condition (dn="(.+,)?<dn>$"). It may differ from one SQL dialect to another (see samples). By default, it is constructed based on the knowledge about how to normalize DN values (e.g. **"<upper_func>(ldap_entries.dn) LIKE CONCAT('%',?)"**); see **upper_func**, **upper_needs_cast**, **concat_pattern** and **strcast_func** in "HELPER CONFIGURATION" for details.

children_cond <SQL expression>

Specifies a where-clause template used to form a children search condition (dn=".+,<dn>$"). It may differ from one SQL dialect to another (see samples). By default, it is constructed based on the knowledge about how to normalize DN values (e.g. **"<upper_func>(ldap_entries.dn) LIKE CONCAT('%,',?)"**); see **upper_func**, **upper_needs_cast**, **concat_pattern** and **strcast_func** in "HELPER CONFIGURATION" for details.

use_subtree_shortcut { YES | no }

Do not use the subtree condition when the searchBase is the database suffix, and the scope is subtree; rather collect all entries.

STATEMENT CONFIGURATION

These options specify SQL query templates for loading schema mapping meta-information, adding and deleting entries to ldap_entries, etc. All these and subtree_cond should have the given default values. For the current value it is recommended to look at the sources, or in the log output when slapd starts with "-d 5" or greater. Note that the parameter number and order must not be changed.

oc_query <SQL expression>

The query that is used to collect the objectClass mapping data from table *ldap_oc_mappings*; see "METAINFORMATION USED" for details. The default is **"SELECT id, name, keytbl, keycol, create_proc, delete_proc, expect_return FROM ldap_oc_mappings"**.

at_query <SQL expression>

The query that is used to collect the attributeType mapping data from table *ldap_attr_mappings*; see "METAINFORMATION USED" for details. The default is **"SELECT name, sel_expr, from_tbls, join_where, add_proc, delete_proc, param_order, expect_return FROM ldap_attr_mappings WHERE oc_map_id=?"**.

id_query <SQL expression>

The query that is used to map a DN to an entry in table *ldap_entries*; see "METAINFORMATION USED" for details. The default is **"SELECT id,keyval,oc_map_id,dn FROM ldap_entries WHERE <DN match expr>"**, where **<DN match expr>** is constructed based on the knowledge about how to normalize DN values (e.g. **"dn=?"** if no means to uppercase strings are available; typically, **"<upper_func>(dn)=?"** is used); see **upper_func**, **upper_needs_cast**, **concat_pattern** and **strcast_func** in "HELPER CONFIGURATION" for details.

insentry_stmt <SQL expression>

The statement that is used to insert a new entry in table *ldap_entries*; see "METAINFORMATION USED" for details. The default is **"INSERT INTO ldap_entries (dn, oc_map_id, parent, keyval) VALUES (?, ?, ?, ?)"**.

delentry_stmt <SQL expression>

The statement that is used to delete an existing entry from table *ldap_entries*; see "METAINFORMATION USED" for details. The default is **"DELETE FROM ldap_entries WHERE id=?"**.

delobjclasses_stmt <SQL expression>

The statement that is used to delete an existing entry's ID from table *ldap_objclasses*; see "METAINFORMATION USED" for details. The default is **"DELETE FROM ldap_entry_objclasses WHERE entry_id=?"**.

HELPER CONFIGURATION

These statements are used to modify the default behavior of the backend according to issues of the dialect of the RDBMS. The first options essentially refer to string and DN normalization when building filters. LDAP normalization is more than upper- (or lower-)casing everything; however, as a reasonable trade-off, for case-sensitive RDBMSes the backend can be instructed to uppercase strings and DNs by providing the **upper_func** directive. Some RDBMSes, to use functions on arbitrary data types, e.g. string constants, requires a cast, which is triggered by the **upper_needs_cast** directive. If required, a string cast function can be provided as well, by using the **strcast_func** directive. Finally, a custom string concatenation pattern may be required; it is provided by the **concat_pattern** directive.

upper_func <SQL function name>

Specifies the name of a function that converts a given value to uppercase. This is used for case insensitive matching when the RDBMS is case sensitive. It may differ from one SQL dialect to another (e.g. **UCASE**, **UPPER** or whatever; see samples). By default, none is used, i.e. strings are not uppercased, so matches may be case sensitive.

upper_needs_cast { NO | yes }

Set this directive to **yes** if **upper_func** needs an explicit cast when applied to literal strings. A cast in the form **CAST (<arg> AS VARCHAR(<max DN length>))** is used, where **<max DN length>** is builtin in back-sql; see macro **BACKSQL_MAX_DN_LEN** (currently 255; note that slapd's builtin limit, in macro **SLAP_LDAPDN_MAXLEN**, is set to 8192). This is *experimental* and may change in future releases.

strcast_func <SQL function name>

Specifies the name of a function that converts a given value to a string for appropriate ordering. This is used in "SELECT DISTINCT" statements for strongly typed RDBMSes with little implicit casting (like PostgreSQL), when a literal string is specified. This is *experimental* and may change in future releases.

concat_pattern <pattern>

This statement defines the **pattern** that is used to concatenate strings. The **pattern** MUST contain two question marks, '?', that will be replaced by the two strings that must be concatenated. The default value is **CONCAT(?,?)**; a form that is known to be highly portable (IBM db2, PostgreSQL) is **?||?**, but an explicit cast may be required when operating on literal strings: **CAST(?||? AS VARCHAR(<length>))**. On some RDBMSes (IBM db2, MSSQL) the form **?+?** is known to work as well. Carefully check the documentation of your RDBMS or stay with the examples for supported ones. This is *experimental* and may change in future releases.

aliasing_keyword <string>

Define the aliasing keyword. Some RDBMSes use the word "*AS*" (the default), others don't use any.

aliasing_quote <string>

Define the quoting char of the aliasing keyword. Some RDBMSes don't require any (the default), others may require single or double quotes.

has_ldapinfo_dn_ru { NO | yes }

Explicitly inform the backend whether the dn_ru column (DN in reverse uppercased form) is present in table *ldap_entries*. Overrides automatic check (this is required, for instance, by PostgreSQL/unixODBC). This is *experimental* and may change in future releases.

fail_if_no_mapping { NO | yes }
When set to **yes** it forces *attribute* write operations to fail if no appropriate mapping between LDAP attributes and SQL data is available. The default behavior is to ignore those changes that cannot be mapped. It has no impact on objectClass mapping, i.e. if the *structuralObjectClass* of an entry cannot be mapped to SQL by looking up its name in ldap_oc_mappings, an *add* operation will fail regardless of the **fail_if_no_mapping** switch; see section "METAINFORMATION USED" for details. This is *experimental* and may change in future releases.

allow_orphans { NO | yes }
When set to **yes** orphaned entries (i.e. without the parent entry in the database) can be added. This option should be used with care, possibly in conjunction with some special rule on the RDBMS side that dynamically creates the missing parent.

baseObject [<filename>]
Instructs the database to create and manage an in-memory baseObject entry instead of looking for one in the RDBMS. If the (optional) **<filename>** argument is given, the entry is read from that file in **LDIF**(5) format; otherwise, an entry with objectClass **extensibleObject** is created based on the contents of the RDN of the *baseObject*. This is particularly useful when *ldap_entries* information is stored in a view rather than in a table, and **union** is not supported for views, so that the view can only specify one rule to compute the entry structure for one objectClass. This topic is discussed further in section "METAINFORMATION USED". This is *experimental* and may change in future releases.

create_needs_select { NO | yes }
Instructs the database whether or not entry creation in table *ldap_entries* needs a subsequent select to collect the automatically assigned ID, instead of being returned by a stored procedure.

fetch_attrs <attrlist>
fetch_all_attrs { NO | yes }
The first statement allows to provide a list of attributes that must always be fetched in addition to those requested by any specific operation, because they are required for the proper usage of the backend. For instance, all attributes used in ACLs should be listed here. The second statement is a shortcut to require all attributes to be always loaded. Note that the dynamically generated attributes, e.g. *hasSubordinates*, *entryDN* and other implementation dependent attributes are **NOT** generated at this point, for consistency with the rest of slapd. This may change in the future.

check_schema { YES | no }
Instructs the database to check schema adherence of entries after modifications, and structural objectClass chain when entries are built. By default it is set to **yes**.

sqllayer <name> [...]
Loads the layer **<name>** onto a stack of helpers that are used to map DNs from LDAP to SQL representation and vice-versa. Subsequent args are passed to the layer configuration routine. This is *highly experimental* and should be used with extreme care. The API of the layers is not frozen yet, so it is unpublished.

METAINFORMATION USED

Almost everything mentioned later is illustrated in examples located in the **servers/slapd/back-sql/rdbms_depend/** directory in the OpenLDAP source tree, and contains scripts for generating sample database for Oracle, MS SQL Server, mySQL and more (including PostgreSQL and IBM db2).

The first thing that one must arrange is what set of LDAP object classes can present your RDBMS

information.

The easiest way is to create an objectClass for each entity you had in ER-diagram when designing your relational schema. Any relational schema, no matter how normalized it is, was designed after some model of your application's domain (for instance, accounts, services etc. in ISP), and is used in terms of its entities, not just tables of normalized schema. It means that for every attribute of every such instance there is an effective SQL query that loads its values.

Also you might want your object classes to conform to some of the standard schemas like inetOrgPerson etc.

Nevertheless, when you think it out, we must define a way to translate LDAP operation requests to (a series of) SQL queries. Let us deal with the SEARCH operation.

Example: Let's suppose that we store information about persons working in our organization in two tables:

```
PERSONS             PHONES
----------          -------------
id integer          id integer
first_name varchar  pers_id integer references persons(id)
last_name varchar   phone
middle_name varchar
...
```

(PHONES contains telephone numbers associated with persons). A person can have several numbers, then PHONES contains several records with corresponding pers_id, or no numbers (and no records in PHONES with such pers_id). An LDAP objectclass to present such information could look like this:

```
person
-------
MUST cn
MAY telephoneNumber $ firstName $ lastName
...
```

To fetch all values for cn attribute given person ID, we construct the query:

```
SELECT CONCAT(persons.first_name,' ',persons.last_name)
   AS cn FROM persons WHERE persons.id=?
```

for telephoneNumber we can use:

```
SELECT phones.phone AS telephoneNumber FROM persons,phones
   WHERE persons.id=phones.pers_id AND persons.id=?
```

If we wanted to service LDAP requests with filters like (telephoneNumber=123*), we would construct something like:

```
SELECT ... FROM persons,phones
   WHERE persons.id=phones.pers_id
      AND persons.id=?
      AND phones.phone like '%1%2%3%'
```

(note how the telephoneNumber match is expanded in multiple wildcards to account for interspersed ininfluential chars like spaces, dashes and so; this occurs by design because telephoneNumber is defined after a specially recognized syntax). So, if we had information about what tables contain values for each attribute, how to join these tables and arrange these values, we could try to automatically generate such statements, and translate search filters to SQL WHERE clauses.

To store such information, we add three more tables to our schema and fill it with data (see samples):

```
ldap_oc_mappings (some columns are not listed for clarity)
---------------
id=1
name="person"
```

keytbl="persons"
keycol="id"

This table defines a mapping between objectclass (its name held in the "name" column), and a table that holds the primary key for corresponding entities. For instance, in our example, the person entity, which we are trying to present as "person" objectclass, resides in two tables (persons and phones), and is identified by the persons.id column (that we will call the primary key for this entity). Keytbl and keycol thus contain "persons" (name of the table), and "id" (name of the column).

ldap_attr_mappings (some columns are not listed for clarity)
---
id=1
oc_map_id=1
name="cn"
sel_expr="CONCAT(persons.first_name,' ',persons.last_name)"
from_tbls="persons"
join_where=NULL
****
id=<n>
oc_map_id=1
name="telephoneNumber"
sel_expr="phones.phone"
from_tbls="persons,phones"
join_where="phones.pers_id=persons.id"

This table defines mappings between LDAP attributes and SQL queries that load their values. Note that, unlike LDAP schema, these are not **attribute types** - the attribute "cn" for "person" objectclass can have its values in different tables than "cn" for some other objectclass, so attribute mappings depend on objectclass mappings (unlike attribute types in LDAP schema, which are indifferent to objectclasses). Thus, we have oc_map_id column with link to oc_mappings table.

Now we cut the SQL query that loads values for a given attribute into 3 parts. First goes into sel_expr column - this is the expression we had between SELECT and FROM keywords, which defines WHAT to load. Next is table list - text between FROM and WHERE keywords. It may contain aliases for convenience (see examples). The last is part of the where clause, which (if it exists at all) expresses the condition for joining the table containing values with the table containing the primary key (foreign key equality and such). If values are in the same table as the primary key, then this column is left NULL (as for cn attribute above).

Having this information in parts, we are able to not only construct queries that load attribute values by id of entry (for this we could store SQL query as a whole), but to construct queries that load id's of objects that correspond to a given search filter (or at least part of it). See below for examples.

ldap_entries
---
id=1
dn=<dn you choose>
oc_map_id=...
parent=<parent record id>
keyval=<value of primary key>

This table defines mappings between DNs of entries in your LDAP tree, and values of primary keys for corresponding relational data. It has recursive structure (parent column references id column of the same table), which allows you to add any tree structure(s) to your flat relational data. Having id of objectclass mapping, we can determine table and column for primary key, and keyval stores value of it, thus defining the exact tuple corresponding to the LDAP entry with this DN.

Note that such design (see exact SQL table creation query) implies one important constraint - the key must be an integer. But all that I know about well-designed schemas makes me think that it's not very narrow ;) If anyone needs support for different types for keys - he may want to write a patch, and submit it to

OpenLDAP ITS, then I'll include it.

Also, several users complained that they don't really need very structured trees, and they don't want to update one more table every time they add or delete an instance in the relational schema. Those people can use a view instead of a real table for ldap_entries, something like this (by Robin Elfrink):

```
CREATE VIEW ldap_entries (id, dn, oc_map_id, parent, keyval)
  AS
    SELECT 0, UPPER('o=MyCompany,c=NL'),
      3, 0, 'baseObject' FROM unixusers WHERE userid='root'
  UNION
    SELECT (1000000000+userid),
      UPPER(CONCAT(CONCAT('cn=',gecos),',o=MyCompany,c=NL')),
      1, 0, userid FROM unixusers
  UNION
    SELECT (2000000000+groupnummer),
      UPPER(CONCAT(CONCAT('cn=',groupnaam),',o=MyCompany,c=NL')),
      2, 0, groupnummer FROM groups;
```

If your RDBMS does not support **unions** in views, only one objectClass can be mapped in **ldap_entries**, and the baseObject cannot be created; in this case, see the **baseObject** directive for a possible workaround.

TYPICAL SQL BACKEND OPERATION

Having meta-information loaded, the SQL backend uses these tables to determine a set of primary keys of candidates (depending on search scope and filter). It tries to do it for each objectclass registered in ldap_objclasses.

Example: for our query with filter (telephoneNumber=123*) we would get the following query generated (which loads candidate IDs)

```
SELECT ldap_entries.id,persons.id, 'person' AS objectClass,
    ldap_entries.dn AS dn
  FROM ldap_entries,persons,phones
 WHERE persons.id=ldap_entries.keyval
   AND ldap_entries.objclass=?
   AND ldap_entries.parent=?
   AND phones.pers_id=persons.id
   AND (phones.phone LIKE '%1%2%3%')
```

(for ONELEVEL search) or "... AND dn=?" (for BASE search) or "... AND dn LIKE '%?'" (for SUBTREE)

Then, for each candidate, we load the requested attributes using per-attribute queries like

```
SELECT phones.phone AS telephoneNumber
  FROM persons,phones
 WHERE persons.id=? AND phones.pers_id=persons.id
```

Then, we use test_filter() from the frontend API to test the entry for a full LDAP search filter match (since we cannot effectively make sense of SYNTAX of corresponding LDAP schema attribute, we translate the filter into the most relaxed SQL condition to filter candidates), and send it to the user.

ADD, DELETE, MODIFY and MODRDN operations are also performed on per-attribute meta-information (add_proc etc.). In those fields one can specify an SQL statement or stored procedure call which can add, or delete given values of a given attribute, using the given entry keyval (see examples -- mostly PostgreSQL, ORACLE and MSSQL - since as of this writing there are no stored procs in MySQL).

We just add more columns to ldap_oc_mappings and ldap_attr_mappings, holding statements to execute (like create_proc, add_proc, del_proc etc.), and flags governing the order of parameters passed to those

statements. Please see samples to find out what are the parameters passed, and other information on this matter - they are self-explanatory for those familiar with the concepts expressed above.

COMMON TECHNIQUES

First of all, let's recall that among other major differences to the complete LDAP data model, the above illustrated concept does not directly support such features as multiple objectclasses per entry, and referrals. Fortunately, they are easy to adopt in this scheme. The SQL backend requires that one more table is added to the schema: ldap_entry_objectclasses(entry_id,oc_name).

That table contains any number of objectclass names that corresponding entries will possess, in addition to that mentioned in mapping. The SQL backend automatically adds attribute mapping for the "objectclass" attribute to each objectclass mapping that loads values from this table. So, you may, for instance, have a mapping for inetOrgPerson, and use it for queries for "person" objectclass...

Referrals used to be implemented in a loose manner by adding an extra table that allowed any entry to host a "ref" attribute, along with a "referral" extra objectClass in table ldap_entry_objclasses. In the current implementation, referrals are treated like any other user-defined schema, since "referral" is a structural objectclass. The suggested practice is to define a "referral" entry in ldap_oc_mappings, holding a naming attribute, e.g. "ou" or "cn", a "ref" attribute, containing the url; in case multiple referrals per entry are needed, a separate table for urls can be created, where urls are mapped to the respective entries. The use of the naming attribute usually requires to add an "extensibleObject" value to ldap_entry_objclasses.

CAVEATS

As previously stated, this backend should not be considered a replacement of other data storage backends, but rather a gateway to existing RDBMS storages that need to be published in LDAP form.

The **hasSubordintes** operational attribute is honored by back-sql in search results and in compare operations; it is partially honored also in filtering. Owing to design limitations, a (brain-dead?) filter of the form **(!(hasSubordinates=TRUE))** will give no results instead of returning all the leaf entries, because it actually expands into **... AND NOT (1=1)**. If you need to find all the leaf entries, please use **(hasSubordinates=FALSE)** instead.

A directoryString value of the form "__First___Last_" (where underscores mean spaces, ASCII 0x20 char) corresponds to its prettified counterpart "First_Last"; this is not currently honored by back-sql if non-prettified data is written via RDBMS; when non-prettified data is written through back-sql, the prettified values are actually used instead.

BUGS

When the **ldap_entry_objclasses** table is empty, filters on the **objectClass** attribute erroneously result in no candidates. A workaround consists in adding at least one row to that table, no matter if valid or not.

PROXY CACHE OVERLAY

The proxy cache overlay allows caching of LDAP search requests (queries) in a local database. See **slapo-pcache**(5) for details.

EXAMPLES

There are example SQL modules in the slapd/back-sql/rdbms_depend/ directory in the OpenLDAP source tree.

ACCESS CONTROL

The **sql** backend honors access control semantics as indicated in **slapd.access**(5) (including the **disclose** access privilege when enabled at compile time).

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8).

NAME

slapd-sql − SQL backend to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The primary purpose of this **slapd**(8) backend is to PRESENT information stored in some RDBMS as an LDAP subtree without any programming (some SQL and maybe stored procedures can't be considered programming, anyway ;).

That is, for example, when you (some ISP) have account information you use in an RDBMS, and want to use modern solutions that expect such information in LDAP (to authenticate users, make email lookups etc.). Or you want to synchronize or distribute information between different sites/applications that use RDBMSes and/or LDAP. Or whatever else...

It is NOT designed as a general-purpose backend that uses RDBMS instead of BerkeleyDB (as the standard BDB backend does), though it can be used as such with several limitations. You can take a look at **http://www.openldap.org/faq/index.cgi?file=378** (OpenLDAP FAQ-O-Matic/General LDAP FAQ/Directories vs. conventional databases) to find out more on this point.

The idea (detailed below) is to use some meta-information to translate LDAP queries to SQL queries, leaving relational schema untouched, so that old applications can continue using it without any modifications. This allows SQL and LDAP applications to inter-operate without replication, and exchange data as needed.

The SQL backend is designed to be tunable to virtually any relational schema without having to change source (through that meta-information mentioned). Also, it uses ODBC to connect to RDBMSes, and is highly configurable for SQL dialects RDBMSes may use, so it may be used for integration and distribution of data on different RDBMSes, OSes, hosts etc., in other words, in highly heterogeneous environment.

This backend is *experimental*.

CONFIGURATION

These **slapd.conf** options apply to the SQL backend database, which means that they must follow a "database sql" line and come before any subsequent "backend" or "database" lines. Other database options not specific to this backend are described in the **slapd.conf**(5) manual page.

DATA SOURCE CONFIGURATION

dbname <datasource name>

The name of the ODBC datasource to use.

dbhost <hostname>
dbpasswd <password>
dbuser <username>

The three above options are generally unneeded, because this information is taken from the datasource specified by the **dbname** directive. They allow to override datasource settings. Also, several RDBMS' drivers tend to require explicit passing of user/password, even if those are given in datasource (Note: **dbhost** is currently ignored).

SCOPING CONFIGURATION

These options specify SQL query templates for scoping searches.

subtree_cond <SQL expression>

Specifies a where-clause template used to form a subtree search condition (dn="(.+,)?<dn>$"). It may differ from one SQL dialect to another (see samples). By default, it is constructed based on the knowledge about how to normalize DN values (e.g. **"<upper_func>(ldap_entries.dn) LIKE CONCAT('%',?)"**); see **upper_func**, **upper_needs_cast**, **concat_pattern** and **strcast_func** in "HELPER CONFIGURATION" for details.

children_cond <SQL expression>

Specifies a where-clause template used to form a children search condition (dn=".+,<dn>$"). It may differ from one SQL dialect to another (see samples). By default, it is constructed based on the knowledge about how to normalize DN values (e.g. **"<upper_func>(ldap_entries.dn) LIKE CONCAT('%,',?)"**); see **upper_func**, **upper_needs_cast**, **concat_pattern** and **strcast_func** in "HELPER CONFIGURATION" for details.

use_subtree_shortcut { YES | no }

Do not use the subtree condition when the searchBase is the database suffix, and the scope is subtree; rather collect all entries.

STATEMENT CONFIGURATION

These options specify SQL query templates for loading schema mapping meta-information, adding and deleting entries to ldap_entries, etc. All these and subtree_cond should have the given default values. For the current value it is recommended to look at the sources, or in the log output when slapd starts with "-d 5" or greater. Note that the parameter number and order must not be changed.

oc_query <SQL expression>

The query that is used to collect the objectClass mapping data from table *ldap_oc_mappings*; see "METAINFORMATION USED" for details. The default is **"SELECT id, name, keytbl, keycol, create_proc, delete_proc, expect_return FROM ldap_oc_mappings"**.

at_query <SQL expression>

The query that is used to collect the attributeType mapping data from table *ldap_attr_mappings*; see "METAINFORMATION USED" for details. The default is **"SELECT name, sel_expr, from_tbls, join_where, add_proc, delete_proc, param_order, expect_return FROM ldap_attr_mappings WHERE oc_map_id=?"**.

id_query <SQL expression>

The query that is used to map a DN to an entry in table *ldap_entries*; see "METAINFORMATION USED" for details. The default is **"SELECT id,keyval,oc_map_id,dn FROM ldap_entries WHERE <DN match expr>"**, where **<DN match expr>** is constructed based on the knowledge about how to normalize DN values (e.g. **"dn=?"** if no means to uppercase strings are available; typically, **"<upper_func>(dn)=?"** is used); see **upper_func**, **upper_needs_cast**, **concat_pattern** and **strcast_func** in "HELPER CONFIGURATION" for details.

insentry_stmt <SQL expression>

The statement that is used to insert a new entry in table *ldap_entries*; see "METAINFORMATION USED" for details. The default is **"INSERT INTO ldap_entries (dn, oc_map_id, parent, keyval) VALUES (?, ?, ?, ?)"**.

delentry_stmt <SQL expression>

The statement that is used to delete an existing entry from table *ldap_entries*; see "METAINFORMATION USED" for details. The default is **"DELETE FROM ldap_entries WHERE id=?"**.

delobjclasses_stmt <SQL expression>

The statement that is used to delete an existing entry's ID from table *ldap_objclasses*; see "METAINFORMATION USED" for details. The default is **"DELETE FROM ldap_entry_objclasses WHERE entry_id=?"**.

HELPER CONFIGURATION

These statements are used to modify the default behavior of the backend according to issues of the dialect of the RDBMS. The first options essentially refer to string and DN normalization when building filters. LDAP normalization is more than upper- (or lower-)casing everything; however, as a reasonable trade-off, for case-sensitive RDBMSes the backend can be instructed to uppercase strings and DNs by providing the **upper_func** directive. Some RDBMSes, to use functions on arbitrary data types, e.g. string constants, requires a cast, which is triggered by the **upper_needs_cast** directive. If required, a string cast function can be provided as well, by using the **strcast_func** directive. Finally, a custom string concatenation pattern may be required; it is provided by the **concat_pattern** directive.

upper_func <SQL function name>

Specifies the name of a function that converts a given value to uppercase. This is used for case insensitive matching when the RDBMS is case sensitive. It may differ from one SQL dialect to another (e.g. **UCASE**, **UPPER** or whatever; see samples). By default, none is used, i.e. strings are not uppercased, so matches may be case sensitive.

upper_needs_cast { NO | yes }

Set this directive to **yes** if **upper_func** needs an explicit cast when applied to literal strings. A cast in the form **CAST (<arg> AS VARCHAR(<max DN length>))** is used, where **<max DN length>** is builtin in back-sql; see macro **BACKSQL_MAX_DN_LEN** (currently 255; note that slapd's builtin limit, in macro **SLAP_LDAPDN_MAXLEN**, is set to 8192). This is *experimental* and may change in future releases.

strcast_func <SQL function name>

Specifies the name of a function that converts a given value to a string for appropriate ordering. This is used in "SELECT DISTINCT" statements for strongly typed RDBMSes with little implicit casting (like PostgreSQL), when a literal string is specified. This is *experimental* and may change in future releases.

concat_pattern <pattern>

This statement defines the **pattern** that is used to concatenate strings. The **pattern** MUST contain two question marks, '?', that will be replaced by the two strings that must be concatenated. The default value is **CONCAT(?,?)**; a form that is known to be highly portable (IBM db2, PostgreSQL) is **?||?**, but an explicit cast may be required when operating on literal strings: **CAST(?||? AS VARCHAR(<length>))**. On some RDBMSes (IBM db2, MSSQL) the form **?+?** is known to work as well. Carefully check the documentation of your RDBMS or stay with the examples for supported ones. This is *experimental* and may change in future releases.

aliasing_keyword <string>

Define the aliasing keyword. Some RDBMSes use the word "*AS*" (the default), others don't use any.

aliasing_quote <string>

Define the quoting char of the aliasing keyword. Some RDBMSes don't require any (the default), others may require single or double quotes.

has_ldapinfo_dn_ru { NO | yes }

Explicitly inform the backend whether the dn_ru column (DN in reverse uppercased form) is present in table *ldap_entries*. Overrides automatic check (this is required, for instance, by PostgreSQL/unixODBC). This is *experimental* and may change in future releases.

fail_if_no_mapping { NO | yes }
When set to **yes** it forces *attribute* write operations to fail if no appropriate mapping between LDAP attributes and SQL data is available. The default behavior is to ignore those changes that cannot be mapped. It has no impact on objectClass mapping, i.e. if the *structuralObjectClass* of an entry cannot be mapped to SQL by looking up its name in ldap_oc_mappings, an *add* operation will fail regardless of the **fail_if_no_mapping** switch; see section "METAINFORMATION USED" for details. This is *experimental* and may change in future releases.

allow_orphans { NO | yes }
When set to **yes** orphaned entries (i.e. without the parent entry in the database) can be added. This option should be used with care, possibly in conjunction with some special rule on the RDBMS side that dynamically creates the missing parent.

baseObject [<filename>]
Instructs the database to create and manage an in-memory baseObject entry instead of looking for one in the RDBMS. If the (optional) **<filename>** argument is given, the entry is read from that file in **LDIF**(5) format; otherwise, an entry with objectClass **extensibleObject** is created based on the contents of the RDN of the *baseObject*. This is particularly useful when *ldap_entries* information is stored in a view rather than in a table, and **union** is not supported for views, so that the view can only specify one rule to compute the entry structure for one objectClass. This topic is discussed further in section "METAINFORMATION USED". This is *experimental* and may change in future releases.

create_needs_select { NO | yes }
Instructs the database whether or not entry creation in table *ldap_entries* needs a subsequent select to collect the automatically assigned ID, instead of being returned by a stored procedure.

fetch_attrs <attrlist>
fetch_all_attrs { NO | yes }
The first statement allows to provide a list of attributes that must always be fetched in addition to those requested by any specific operation, because they are required for the proper usage of the backend. For instance, all attributes used in ACLs should be listed here. The second statement is a shortcut to require all attributes to be always loaded. Note that the dynamically generated attributes, e.g. *hasSubordinates*, *entryDN* and other implementation dependent attributes are **NOT** generated at this point, for consistency with the rest of slapd. This may change in the future.

check_schema { YES | no }
Instructs the database to check schema adherence of entries after modifications, and structural objectClass chain when entries are built. By default it is set to **yes**.

sqllayer <name> [...]
Loads the layer **<name>** onto a stack of helpers that are used to map DNs from LDAP to SQL representation and vice-versa. Subsequent args are passed to the layer configuration routine. This is *highly experimental* and should be used with extreme care. The API of the layers is not frozen yet, so it is unpublished.

METAINFORMATION USED

Almost everything mentioned later is illustrated in examples located in the **servers/slapd/back-sql/rdbms_depend/** directory in the OpenLDAP source tree, and contains scripts for generating sample database for Oracle, MS SQL Server, mySQL and more (including PostgreSQL and IBM db2).

The first thing that one must arrange is what set of LDAP object classes can present your RDBMS

information.

The easiest way is to create an objectClass for each entity you had in ER-diagram when designing your relational schema. Any relational schema, no matter how normalized it is, was designed after some model of your application's domain (for instance, accounts, services etc. in ISP), and is used in terms of its entities, not just tables of normalized schema. It means that for every attribute of every such instance there is an effective SQL query that loads its values.

Also you might want your object classes to conform to some of the standard schemas like inetOrgPerson etc.

Nevertheless, when you think it out, we must define a way to translate LDAP operation requests to (a series of) SQL queries. Let us deal with the SEARCH operation.

Example: Let's suppose that we store information about persons working in our organization in two tables:

```
PERSONS             PHONES
----------          -------------
id integer          id integer
first_name varchar  pers_id integer references persons(id)
last_name varchar   phone
middle_name varchar
...
```

(PHONES contains telephone numbers associated with persons). A person can have several numbers, then PHONES contains several records with corresponding pers_id, or no numbers (and no records in PHONES with such pers_id). An LDAP objectclass to present such information could look like this:

```
person
-------
MUST cn
MAY telephoneNumber $ firstName $ lastName
...
```

To fetch all values for cn attribute given person ID, we construct the query:

```
SELECT CONCAT(persons.first_name,' ',persons.last_name)
   AS cn FROM persons WHERE persons.id=?
```

for telephoneNumber we can use:

```
SELECT phones.phone AS telephoneNumber FROM persons,phones
   WHERE persons.id=phones.pers_id AND persons.id=?
```

If we wanted to service LDAP requests with filters like (telephoneNumber=123*), we would construct something like:

```
SELECT ... FROM persons,phones
   WHERE persons.id=phones.pers_id
      AND persons.id=?
      AND phones.phone like '%1%2%3%'
```

(note how the telephoneNumber match is expanded in multiple wildcards to account for interspersed ininfluential chars like spaces, dashes and so; this occurs by design because telephoneNumber is defined after a specially recognized syntax). So, if we had information about what tables contain values for each attribute, how to join these tables and arrange these values, we could try to automatically generate such statements, and translate search filters to SQL WHERE clauses.

To store such information, we add three more tables to our schema and fill it with data (see samples):

```
ldap_oc_mappings (some columns are not listed for clarity)
---------------
id=1
name="person"
```

keytbl="persons"
keycol="id"

This table defines a mapping between objectclass (its name held in the "name" column), and a table that holds the primary key for corresponding entities. For instance, in our example, the person entity, which we are trying to present as "person" objectclass, resides in two tables (persons and phones), and is identified by the persons.id column (that we will call the primary key for this entity). Keytbl and keycol thus contain "persons" (name of the table), and "id" (name of the column).

ldap_attr_mappings (some columns are not listed for clarity)
---
id=1
oc_map_id=1
name="cn"
sel_expr="CONCAT(persons.first_name,' ',persons.last_name)"
from_tbls="persons"
join_where=NULL
****
id=<n>
oc_map_id=1
name="telephoneNumber"
sel_expr="phones.phone"
from_tbls="persons,phones"
join_where="phones.pers_id=persons.id"

This table defines mappings between LDAP attributes and SQL queries that load their values. Note that, unlike LDAP schema, these are not **attribute types** - the attribute "cn" for "person" objectclass can have its values in different tables than "cn" for some other objectclass, so attribute mappings depend on objectclass mappings (unlike attribute types in LDAP schema, which are indifferent to objectclasses). Thus, we have oc_map_id column with link to oc_mappings table.

Now we cut the SQL query that loads values for a given attribute into 3 parts. First goes into sel_expr column - this is the expression we had between SELECT and FROM keywords, which defines WHAT to load. Next is table list - text between FROM and WHERE keywords. It may contain aliases for convenience (see examples). The last is part of the where clause, which (if it exists at all) expresses the condition for joining the table containing values with the table containing the primary key (foreign key equality and such). If values are in the same table as the primary key, then this column is left NULL (as for cn attribute above).

Having this information in parts, we are able to not only construct queries that load attribute values by id of entry (for this we could store SQL query as a whole), but to construct queries that load id's of objects that correspond to a given search filter (or at least part of it). See below for examples.

ldap_entries
---
id=1
dn=<dn you choose>
oc_map_id=...
parent=<parent record id>
keyval=<value of primary key>

This table defines mappings between DNs of entries in your LDAP tree, and values of primary keys for corresponding relational data. It has recursive structure (parent column references id column of the same table), which allows you to add any tree structure(s) to your flat relational data. Having id of objectclass mapping, we can determine table and column for primary key, and keyval stores value of it, thus defining the exact tuple corresponding to the LDAP entry with this DN.

Note that such design (see exact SQL table creation query) implies one important constraint - the key must be an integer. But all that I know about well-designed schemas makes me think that it's not very narrow ;) If anyone needs support for different types for keys - he may want to write a patch, and submit it to

OpenLDAP ITS, then I'll include it.

Also, several users complained that they don't really need very structured trees, and they don't want to update one more table every time they add or delete an instance in the relational schema. Those people can use a view instead of a real table for ldap_entries, something like this (by Robin Elfrink):

```
CREATE VIEW ldap_entries (id, dn, oc_map_id, parent, keyval)
  AS
    SELECT 0, UPPER('o=MyCompany,c=NL'),
      3, 0, 'baseObject' FROM unixusers WHERE userid='root'
  UNION
    SELECT (1000000000+userid),
      UPPER(CONCAT(CONCAT('cn=',gecos),',o=MyCompany,c=NL')),
      1, 0, userid FROM unixusers
  UNION
    SELECT (2000000000+groupnummer),
      UPPER(CONCAT(CONCAT('cn=',groupnaam),',o=MyCompany,c=NL')),
      2, 0, groupnummer FROM groups;
```

If your RDBMS does not support **unions** in views, only one objectClass can be mapped in **ldap_entries**, and the baseObject cannot be created; in this case, see the **baseObject** directive for a possible workaround.

TYPICAL SQL BACKEND OPERATION

Having meta-information loaded, the SQL backend uses these tables to determine a set of primary keys of candidates (depending on search scope and filter). It tries to do it for each objectclass registered in ldap_objclasses.

Example: for our query with filter (telephoneNumber=123*) we would get the following query generated (which loads candidate IDs)

```
SELECT ldap_entries.id,persons.id, 'person' AS objectClass,
    ldap_entries.dn AS dn
  FROM ldap_entries,persons,phones
 WHERE persons.id=ldap_entries.keyval
   AND ldap_entries.objclass=?
   AND ldap_entries.parent=?
   AND phones.pers_id=persons.id
   AND (phones.phone LIKE '%1%2%3%')
```

(for ONELEVEL search) or "... AND dn=?" (for BASE search) or "... AND dn LIKE '%?'" (for SUBTREE)

Then, for each candidate, we load the requested attributes using per-attribute queries like

```
SELECT phones.phone AS telephoneNumber
  FROM persons,phones
 WHERE persons.id=? AND phones.pers_id=persons.id
```

Then, we use test_filter() from the frontend API to test the entry for a full LDAP search filter match (since we cannot effectively make sense of SYNTAX of corresponding LDAP schema attribute, we translate the filter into the most relaxed SQL condition to filter candidates), and send it to the user.

ADD, DELETE, MODIFY and MODRDN operations are also performed on per-attribute meta-information (add_proc etc.). In those fields one can specify an SQL statement or stored procedure call which can add, or delete given values of a given attribute, using the given entry keyval (see examples -- mostly PostgreSQL, ORACLE and MSSQL - since as of this writing there are no stored procs in MySQL).

We just add more columns to ldap_oc_mappings and ldap_attr_mappings, holding statements to execute (like create_proc, add_proc, del_proc etc.), and flags governing the order of parameters passed to those

statements. Please see samples to find out what are the parameters passed, and other information on this matter - they are self-explanatory for those familiar with the concepts expressed above.

COMMON TECHNIQUES

First of all, let's recall that among other major differences to the complete LDAP data model, the above illustrated concept does not directly support such features as multiple objectclasses per entry, and referrals. Fortunately, they are easy to adopt in this scheme. The SQL backend requires that one more table is added to the schema: ldap_entry_objectclasses(entry_id,oc_name).

That table contains any number of objectclass names that corresponding entries will possess, in addition to that mentioned in mapping. The SQL backend automatically adds attribute mapping for the "objectclass" attribute to each objectclass mapping that loads values from this table. So, you may, for instance, have a mapping for inetOrgPerson, and use it for queries for "person" objectclass...

Referrals used to be implemented in a loose manner by adding an extra table that allowed any entry to host a "ref" attribute, along with a "referral" extra objectClass in table ldap_entry_objclasses. In the current implementation, referrals are treated like any other user-defined schema, since "referral" is a structural objectclass. The suggested practice is to define a "referral" entry in ldap_oc_mappings, holding a naming attribute, e.g. "ou" or "cn", a "ref" attribute, containing the url; in case multiple referrals per entry are needed, a separate table for urls can be created, where urls are mapped to the respective entries. The use of the naming attribute usually requires to add an "extensibleObject" value to ldap_entry_objclasses.

CAVEATS

As previously stated, this backend should not be considered a replacement of other data storage backends, but rather a gateway to existing RDBMS storages that need to be published in LDAP form.

The **hasSubordintes** operational attribute is honored by back-sql in search results and in compare operations; it is partially honored also in filtering. Owing to design limitations, a (brain-dead?) filter of the form **(!(hasSubordinates=TRUE))** will give no results instead of returning all the leaf entries, because it actually expands into **... AND NOT (1=1)**. If you need to find all the leaf entries, please use **(hasSubordinates=FALSE)** instead.

A directoryString value of the form "__First___Last_" (where underscores mean spaces, ASCII 0x20 char) corresponds to its prettified counterpart "First_Last"; this is not currently honored by back-sql if non-prettified data is written via RDBMS; when non-prettified data is written through back-sql, the prettified values are actually used instead.

BUGS

When the **ldap_entry_objclasses** table is empty, filters on the **objectClass** attribute erroneously result in no candidates. A workaround consists in adding at least one row to that table, no matter if valid or not.

PROXY CACHE OVERLAY

The proxy cache overlay allows caching of LDAP search requests (queries) in a local database. See **slapo-pcache**(5) for details.

EXAMPLES

There are example SQL modules in the slapd/back-sql/rdbms_depend/ directory in the OpenLDAP source tree.

ACCESS CONTROL

The **sql** backend honors access control semantics as indicated in **slapd.access**(5) (including the **disclose** access privilege when enabled at compile time).

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8).

NAME

slapd.access − access configuration for slapd, the stand-alone LDAP daemon

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **slapd.conf**(5) file contains configuration information for the **slapd**(8) daemon. This configuration file is also used by the SLAPD tools **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), and **slaptest**(8).

The **slapd.conf** file consists of a series of global configuration options that apply to **slapd** as a whole (including all backends), followed by zero or more database backend definitions that contain information specific to a backend instance.

The general format of **slapd.conf** is as follows:

```
# comment - these options apply to every database
<global configuration options>
# first database definition & configuration options
database    <backend 1 type>
<configuration options specific to backend 1>
# subsequent database definitions & configuration options
...
```

Both the global configuration and each backend-specific section can contain access information. Backend-specific access control directives are used for those entries that belong to the backend, according to their naming context. In case no access control directives are defined for a backend or those which are defined are not applicable, the directives from the global configuration section are then used.

If no access controls are present, the default policy allows anyone and everyone to read anything but restricts updates to rootdn. (e.g., "access to * by * read"). The rootdn can always read and write EVERYTHING!

For entries not held in any backend (such as a root DSE), the directives of the first backend (and any global directives) are used.

Arguments that should be replaced by actual text are shown in brackets <>.

THE ACCESS DIRECTIVE

The structure of the access control directives is

access to <what> [by <who> [<access>] [<control>]]+

Grant access (specified by **<access>**) to a set of entries and/or attributes (specified by **<what>**) by one or more requestors (specified by **<who>**).

Lists of access directives are evaluated in the order they appear in *slapd.conf*. When a **<what>** clause matches the datum whose access is being evaluated, its **<who>** clause list is checked. When a **<who>** clause matches the accessor's properties, its **<access>** and **<control>** clauses are evaluated. Access control checking stops at the first match of the **<what>** and **<who>** clause, unless otherwise dictated by the **<control>** clause. Each **<who>** clause list is implicitly terminated by a

```
by * none stop
```

clause that results in stopping the access control with no access privileges granted. Each **<what>** clause list is implicitly terminated by a

```
access to *
        by * none
```

clause that results in granting no access privileges to an otherwise unspecified datum.

THE <WHAT> FIELD

The field **<what>** specifies the entity the access control directive applies to. It can have the forms

dn[.<dnstyle>]=<dnpattern>
filter=<ldapfilter>
attrs=<attrlist>[val[/matchingRule][.<attrstyle>]=<attrval>]

with

<dnstyle>={{exact|base(object)}|regex
|one(level)|sub(tree)|children}
<attrlist>={<attr>|[{!|@}]<objectClass>}[,<attrlist>]
<attrstyle>={{exact|base(object)}|regex
|one(level)|sub(tree)|children}

The statement **dn=<dnpattern>** selects the entries based on their naming context. The **<dnpattern>** is a string representation of the entry's DN. The wildcard ***** stands for all the entries, and it is implied if no **dn** form is given.

The **<dnstyle>** is optional; however, it is recommended to specify it to avoid ambiguities. **Base** (synonym of **baseObject**), the default, or **exact** (an alias of **base**) indicates the entry whose DN is equal to the **<dnpattern>**; **one** (synonym of **onelevel**) indicates all the entries immediately below the **<dnpattern>**, **sub** (synonym of **subtree**) indicates all entries in the subtree at the **<dnpattern>**, **children** indicates all the entries below (subordinate to) the **<dnpattern>**.

If the **<dnstyle>** qualifier is **regex**, then **<dnpattern>** is a POSIX (''extended'') regular expression pattern, as detailed in **regex**(7) and/or **re_format**(7), matching a normalized string representation of the entry's DN. The regex form of the pattern does not (yet) support UTF−8.

The statement **filter=<ldapfilter>** selects the entries based on a valid LDAP filter as described in RFC 4515. A filter of **(objectClass=*)** is implied if no **filter** form is given.

The statement **attrs=<attrlist>** selects the attributes the access control rule applies to. It is a comma-separated list of attribute types, plus the special names **entry**, indicating access to the entry itself, and **children**, indicating access to the entry's children. ObjectClass names may also be specified in this list, which will affect all the attributes that are required and/or allowed by that objectClass. Actually, names in **<attrlist>** that are prefixed by **@** are directly treated as objectClass names. A name prefixed by **!** is also treated as an objectClass, but in this case the access rule affects the attributes that are not required nor allowed by that objectClass. If no **attrs** form is given, **attrs=@extensibleObject** is implied, i.e. all attributes are addressed.

Using the form **attrs=<attr> val[/matchingRule][.<attrstyle>]=<attrval>** specifies access to a particular value of a single attribute. In this case, only a single attribute type may be given. The **<attrstyle>** **exact** (the default) uses the attribute's equality matching rule to compare the value, unless a different (and compatible) matching rule is specified. If the **<attrstyle>** is **regex**, the provided value is used as a POSIX (''extended'') regular expression pattern. If the attribute has DN syntax, the **<attrstyle>** can be any of **base**, **onelevel**, **subtree** or **children**, resulting in base, onelevel, subtree or children match, respectively.

The dn, filter, and attrs statements are additive; they can be used in sequence to select entities the access rule applies to based on naming context, value and attribute type simultaneously.

THE <WHO> FIELD

The field **<who>** indicates whom the access rules apply to. Multiple **<who>** statements can appear in an access control statement, indicating the different access privileges to the same resource that apply to different accessee. It can have the forms

*
anonymous
users
self[.<selfstyle>]

dn[.<dnstyle>[,<modifier>]]=<DN>

dnattr=<attrname>

realanonymous
realusers
realself[.<selfstyle>]

realdn[.<dnstyle>[,<modifier>]]=<DN>
realdnattr=<attrname>

group[/<objectclass>[/<attrname>]]
[.<groupstyle>]=<group>
peername[.<peernamestyle>]=<peername>
sockname[.<style>]=<sockname>
domain[.<domainstyle>[,<modifier>]]=<domain>
sockurl[.<style>]=<sockurl>
set[.<setstyle>]=<pattern>

ssf=<n>
transport_ssf=<n>
tls_ssf=<n>
sasl_ssf=<n>

dynacl/<name>[/<options>][.<dynstyle>][=<pattern>]

with

<style>={exact|regex|expand}
<selfstyle>={level{<n>}}
<dnstyle>={{exact|base(object)}|regex
|one(level)|sub(tree)|children|level{<n>}}
<groupstyle>={exact|expand}
<peernamestyle>={<style>|ip|ipv6|path}
<domainstyle>={exact|regex|sub(tree)}
<setstyle>={exact|regex}
<modifier>={expand}
<name>=aci <pattern>=<attrname>]

They may be specified in combination.

The wildcard ***** refers to everybody.

The keywords prefixed by **real** act as their counterparts without prefix; the checking respectively occurs with the *authentication* DN and the *authorization* DN.

The keyword **anonymous** means access is granted to unauthenticated clients; it is mostly used to limit access to authentication resources (e.g. the **userPassword** attribute) to unauthenticated clients for authentication purposes.

The keyword **users** means access is granted to authenticated clients.

The keyword **self** means access to an entry is allowed to the entry itself (e.g. the entry being accessed and the requesting entry must be the same). It allows the **level{<n>}** style, where <*n*> indicates what ancestor of the DN is to be used in matches. A positive value indicates that the <n>-th ancestor of the user's DN is to be considered; a negative value indicates that the <n>-th ancestor of the target is to be considered. For example, a "*by self.level{1} ...*" clause would match when the object "*dc=example,dc=com*" is accessed by "*cn=User,dc=example,dc=com*". A "*by self.level{-1} ...*" clause would match when the same user accesses the object "*ou=Address Book,cn=User,dc=example,dc=com*".

The statement **dn=<DN>** means that access is granted to the matching DN. The optional style qualifier

dnstyle allows the same choices of the dn form of the **<what>** field. In addition, the **regex** style can exploit substring substitution of submatches in the **<what>** dn.regex clause by using the form **$<digit>**, with **digit** ranging from 0 to 9 (where 0 matches the entire string), or the form **${<digit>+}**, for submatches higher than 9. Since the dollar character is used to indicate a substring replacement, the dollar character that is used to indicate match up to the end of the string must be escaped by a second dollar character, e.g.

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=[^,]+,dc=com$"
    by dn.regex="^uid=$2,dc=[^,]+,dc=com$$" write
```

The style qualifier allows an optional **modifier**. At present, the only type allowed is **expand**, which causes substring substitution of submatches to take place even if **dnstyle** is not **regex**. Note that the **regex** dnstyle in the above example may be of use only if the **<by>** clause needs to be a regex; otherwise, if the value of the second (from the right) **dc=** portion of the DN in the above example were fixed, the form

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
    by dn.exact,expand="uid=$2,dc=example,dc=com" write
```

could be used; if it had to match the value in the **<what>** clause, the form

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=([^,]+),dc=com$"
    by dn.exact,expand="uid=$2,dc=$3,dc=com" write
```

could be used.

Forms of the **<what>** clause other than regex may provide submatches as well. The **base(object)**, the **sub(tree)**, the **one(level)**, and the **children** forms provide **$0** as the match of the entire string. The **sub(tree)**, the **one(level)**, and the **children** forms also provide **$1** as the match of the rightmost part of the DN as defined in the **<what>** clause. This may be useful, for instance, to provide access to all the ancestors of a user by defining

```
access to dn.subtree="dc=com"
    by dn.subtree,expand="$1" read
```

which means that only access to entries that appear in the DN of the **<by>** clause is allowed.

The **level{<n>}** form is an extension and a generalization of the **onelevel** form, which matches all DNs whose <n>-th ancestor is the pattern. So, *level{1}* is equivalent to *onelevel*, and *level{0}* is equivalent to *base*.

It is perfectly useless to give any access privileges to a DN that exactly matches the **rootdn** of the database the ACLs apply to, because it implicitly possesses write privileges for the entire tree of that database. Actually, access control is bypassed for the **rootdn**, to solve the intrinsic chicken-and-egg problem.

The statement **dnattr=<attrname>** means that access is granted to requests whose DN is listed in the entry being accessed under the **<attrname>** attribute.

The statement **group=<group>** means that access is granted to requests whose DN is listed in the group entry whose DN is given by **<group>**. The optional parameters **<objectclass>** and **<attrname>** define the objectClass and the member attributeType of the group entry. The defaults are **groupOfNames** and **member**, respectively. The optional style qualifier **<style>** can be **expand**, which means that **<group>** will be expanded as a replacement string (but not as a regular expression) according to **regex**(7) and/or **re_format**(7), and **exact**, which means that exact match will be used. If the style of the DN portion of the **<what>** clause is regex, the submatches are made available according to **regex**(7) and/or **re_format**(7); other styles provide limited submatches as discussed above about the DN form of the **<by>** clause.

For static groups, the specified attributeType must have **DistinguishedName** or **NameAndOptionalUID** syntax. For dynamic groups the attributeType must be a subtype of the **labeledURI** attributeType. Only LDAP URIs of the form **ldap:///<base>??<scope>?<filter>** will be evaluated in a dynamic group, by searching the local server only.

The statements **peername=<peername>**, **sockname=<sockname>**, **domain=<domain>**, and **sockurl=<sockurl>** mean that the contacting host IP (in the form **IP=<ip>:<port>** for IPv4, or **IP=[<ipv6>]:<port>** for IPv6) or the contacting host named pipe file name (in the form **PATH=<path>** if

connecting through a named pipe) for **peername**, the named pipe file name for **sockname**, the contacting host name for **domain**, and the contacting URL for **sockurl** are compared against **pattern** to determine access. The same **style** rules for pattern match described for the **group** case apply, plus the **regex** style, which implies submatch **expand** and regex match of the corresponding connection parameters. The **exact** style of the **<peername>** clause (the default) implies a case-exact match on the client's **IP**, including the **IP=** prefix and the trailing **:<port>**, or the client's **path**, including the **PATH=** prefix if connecting through a named pipe. The special **ip** style interprets the pattern as **<peername>=<ip>[%<mask>][{<n>}]**, where **<ip>** and **<mask>** are dotted digit representations of the IP and the mask, while **<n>**, delimited by curly brackets, is an optional port. The same applies to IPv6 addresses when the special **ipv6** style is used. When checking access privileges, the IP portion of the **peername** is extracted, eliminating the **IP=** prefix and the **:<port>** part, and it is compared against the **<ip>** portion of the pattern after masking with **<mask>**: *((peername & <mask>) == <ip>)*. As an example, **peername.ip=127.0.0.1** and **peername.ipv6=::1** allow connections only from localhost, **peername.ip=192.168.1.0%255.255.255.0** allows connections from any IP in the 192.168.1 class C domain, and **peername.ip=192.168.1.16%255.255.255.240{9009}** allows connections from any IP in the 192.168.1.[16-31] range of the same domain, only if port 9009 is used. The special **path** style eliminates the **PATH=** prefix from the **peername** when connecting through a named pipe, and performs an exact match on the given pattern. The **<domain>** clause also allows the **subtree** style, which succeeds when a fully qualified name exactly matches the **domain** pattern, or its trailing part, after a **dot**, exactly matches the **domain** pattern. The **expand** style is allowed, implying an **exact** match with submatch expansion; the use of **expand** as a style modifier is considered more appropriate. As an example, **domain.subtree=example.com** will match www.example.com, but will not match www.anotherexample.com. The **domain** of the contacting host is determined by performing a DNS reverse lookup. As this lookup can easily be spoofed, use of the **domain** statement is strongly discouraged. By default, reverse lookups are disabled. The optional **domainstyle** qualifier of the **<domain>** clause allows a **modifier** option; the only value currently supported is **expand**, which causes substring substitution of submatches to take place even if the **domainstyle** is not **regex**, much like the analogous usage in **<dn>** clause.

The statement **set=<pattern>** is undocumented yet.

The statement **dynacl/<name>[/<options>][.<dynstyle>][=<pattern>]** means that access checking is delegated to the admin-defined method indicated by **<name>**, which can be registered at run-time by means of the **moduleload** statement. The fields **<options>**, **<dynstyle>** and **<pattern>** are optional, and are directly passed to the registered parsing routine. Dynacl is experimental; it must be enabled at compile time.

The statement **dynacl/aci[=<attrname>]** means that the access control is determined by the values in the **attrname** of the entry itself. The optional **<attrname>** indicates what attributeType holds the ACI information in the entry. By default, the **OpenLDAPaci** operational attribute is used. ACIs are experimental; they must be enabled at compile time.

The statements **ssf=<n>**, **transport_ssf=<n>**, **tls_ssf=<n>**, and **sasl_ssf=<n>** set the minimum required Security Strength Factor (ssf) needed to grant access. The value should be positive integer.

THE <ACCESS> FIELD

The optional field **<access> ::= [[real]self]{<level>|<priv>}** determines the access level or the specific access privileges the **who** field will have. Its component are defined as

<level> ::= none|disclose|auth|compare|search|read|write|manage
<priv> ::= {=|+|-}{m|w|r|s|c|x|d|0}+

The modifier **self** allows special operations like having a certain access level or privilege only in case the operation involves the name of the user that's requesting the access. It implies the user that requests access is authorized. The modifier **realself** refers to the authenticated DN as opposed to the authorized DN of the **self** modifier. An example is the **selfwrite** access to the member attribute of a group, which allows one to add/delete its own DN from the member list of a group, without affecting other members.

The **level** access model relies on an incremental interpretation of the access privileges. The possible levels are **none**, **disclose**, **auth**, **compare**, **search**, **read**, and **write**. Each access level implies all the preceding ones, thus **manage** grants all access including administrative access,

The **none** access level disallows all access including disclosure on error.

The **disclose** access level allows disclosure of information on error.

The **auth** access level means that one is allowed access to an attribute to perform authentication/authorization operations (e.g. **bind**) with no other access. This is useful to grant unauthenticated clients the least possible access level to critical resources, like passwords.

The **priv** access model relies on the explicit setting of access privileges for each clause. The **=** sign resets previously defined accesses; as a consequence, the final access privileges will be only those defined by the clause. The **+** and **-** signs add/remove access privileges to the existing ones. The privileges are **m** for manage, **w** for write, **r** for read, **s** for search, **c** for compare, **x** for authentication, and **d** for disclose. More than one of the above privileges can be added in one statement. **0** indicates no privileges and is used only by itself (e.g., +0).

If no access is given, it defaults to **+0**.

THE <CONTROL> FIELD

The optional field **<control>** controls the flow of access rule application. It can have the forms

```
stop
continue
break
```

where **stop**, the default, means access checking stops in case of match. The other two forms are used to keep on processing access clauses. In detail, the **continue** form allows for other **<who>** clauses in the same **<access>** clause to be considered, so that they may result in incrementally altering the privileges, while the **break** form allows for other **<access>** clauses that match the same target to be processed. Consider the (silly) example

```
access to dn.subtree="dc=example,dc=com" attrs=cn
        by * =cs break

access to dn.subtree="ou=People,dc=example,dc=com"
        by * +r
```

which allows search and compare privileges to everybody under the "dc=example,dc=com" tree, with the second rule allowing also read in the "ou=People" subtree, or the (even more silly) example

```
access to dn.subtree="dc=example,dc=com" attrs=cn
        by * =cs continue
        by users +r
```

which grants everybody search and compare privileges, and adds read privileges to authenticated clients.

One useful application is to easily grant write privileges to an **updatedn** that is different from the **rootdn**. In this case, since the **updatedn** needs write access to (almost) all data, one can use

```
access to *
        by dn.exact="cn=The Update DN,dc=example,dc=com" write
        by * break
```

as the first access rule. As a consequence, unless the operation is performed with the **updatedn** identity, control is passed straight to the subsequent rules.

OPERATION REQUIREMENTS

Operations require different privileges on different portions of entries. The following summary applies to primary database backends such as the BDB and HDB backends. Requirements for other backends may (and often do) differ.

The **add** operation requires **write (=w)** privileges on the pseudo-attribute **entry** of the entry being added, and **write (=w)** privileges on the pseudo-attribute **children** of the entry's parent. When adding the suffix

entry of a database, write access to **children** of the empty DN ("") is required.

The **bind** operation, when credentials are stored in the directory, requires **auth (=x)** privileges on the attribute the credentials are stored in (usually **userPassword**).

The **compare** operation requires **compare (=c)** privileges on the attribute that is being compared.

The **delete** operation requires **write (=w)** privileges on the pseudo-attribute **entry** of the entry being deleted, and **write (=w)** privileges on the **children** pseudo-attribute of the entry's parent.

The **modify** operation requires **write (=w)** privileges on the attributes being modified.

The **modrdn** operation requires **write (=w)** privileges on the pseudo-attribute **entry** of the entry whose relative DN is being modified, **write (=w)** privileges on the pseudo-attribute **children** of the old and new entry's parents, and **write (=w)** privileges on the attributes that are present in the new relative DN. **Write (=w)** privileges are also required on the attributes that are present in the old relative DN if **deleteoldrdn** is set to 1.

The **search** operation, requires **search (=s)** privileges on the **entry** pseudo-attribute of the searchBase (NOTE: this was introduced with OpenLDAP 2.4). Then, for each entry, it requires **search (=s)** privileges on the attributes that are defined in the filter. The resulting entries are finally tested for **read (=r)** privileges on the pseudo-attribute **entry** (for read access to the entry itself) and for **read (=r)** access on each value of each attribute that is requested. Also, for each **referral** object used in generating continuation references, the operation requires **read (=r)** access on the pseudo-attribute **entry** (for read access to the referral object itself), as well as **read (=r)** access to the attribute holding the referral information (generally the **ref** attribute).

Some internal operations and some **controls** require specific access privileges. The **authzID** mapping and the **proxyAuthz** control require **auth (=x)** privileges on all the attributes that are present in the search filter of the URI regexp maps (the right-hand side of the **authz-regexp** directives). **Auth (=x)** privileges are also required on the **authzTo** attribute of the authorizing identity and/or on the **authzFrom** attribute of the authorized identity. In general, when an internal lookup is performed for authentication or authorization purposes, search-specific privileges (see the access requirements for the search operation illustrated above) are relaxed to **auth**.

Access control to search entries is checked by the frontend, so it is fully honored by all backends; for all other operations and for the discovery phase of the search operation, full ACL semantics is only supported by the primary backends, i.e. **back-bdb**(5), and **back-hdb**(5).

Some other backend, like **back-sql**(5), may fully support them; others may only support a portion of the described semantics, or even differ in some aspects. The relevant details are described in the backend-specific man pages.

CAVEATS

It is strongly recommended to explicitly use the most appropriate **<dnstyle>** in **<what>** and **<who>** clauses, to avoid possible incorrect specifications of the access rules as well as for performance (avoid unnecessary regex matching when an exact match suffices) reasons.

An administrator might create a rule of the form:

```
access to dn.regex="dc=example,dc=com"
        by ...
```

expecting it to match all entries in the subtree "dc=example,dc=com". However, this rule actually matches any DN which contains anywhere the substring "dc=example,dc=com". That is, the rule matches both "uid=joe,dc=example,dc=com" and "dc=example,dc=com,uid=joe".

To match the desired subtree, the rule would be more precisely written:

```
access to dn.regex="^(.+,)?dc=example,dc=com$"
        by ...
```

For performance reasons, it would be better to use the subtree style.

```
access to dn.subtree="dc=example,dc=com"
        by ...
```

When writing submatch rules, it may be convenient to avoid unnecessary **regex <dnstyle>** use; for instance, to allow access to the subtree of the user that matches the **<what>** clause, one could use

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
        by dn.regex="^uid=$2,dc=example,dc=com$$" write
        by ...
```

However, since all that is required in the **<by>** clause is substring expansion, a more efficient solution is

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
        by dn.exact,expand="uid=$2,dc=example,dc=com" write
        by ...
```

In fact, while a **<dnstyle>** of **regex** implies substring expansion, **exact**, as well as all the other DN specific **<dnstyle>** values, does not, so it must be explicitly requested.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd(8), **slapd-***(5), **slapacl**(8), **regex**(7), **re_format**(7)

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

NAME

slapd.access − access configuration for slapd, the stand-alone LDAP daemon

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **slapd.conf**(5) file contains configuration information for the **slapd**(8) daemon. This configuration file is also used by the SLAPD tools **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), and **slaptest**(8).

The **slapd.conf** file consists of a series of global configuration options that apply to **slapd** as a whole (including all backends), followed by zero or more database backend definitions that contain information specific to a backend instance.

The general format of **slapd.conf** is as follows:

```
# comment - these options apply to every database
<global configuration options>
# first database definition & configuration options
database    <backend 1 type>
<configuration options specific to backend 1>
# subsequent database definitions & configuration options
...
```

Both the global configuration and each backend-specific section can contain access information. Backend-specific access control directives are used for those entries that belong to the backend, according to their naming context. In case no access control directives are defined for a backend or those which are defined are not applicable, the directives from the global configuration section are then used.

If no access controls are present, the default policy allows anyone and everyone to read anything but restricts updates to rootdn. (e.g., "access to * by * read"). The rootdn can always read and write EVERYTHING!

For entries not held in any backend (such as a root DSE), the directives of the first backend (and any global directives) are used.

Arguments that should be replaced by actual text are shown in brackets <>.

THE ACCESS DIRECTIVE

The structure of the access control directives is

access to <what> [by <who> [<access>] [<control>]]+

Grant access (specified by **<access>**) to a set of entries and/or attributes (specified by **<what>**) by one or more requestors (specified by **<who>**).

Lists of access directives are evaluated in the order they appear in *slapd.conf*. When a **<what>** clause matches the datum whose access is being evaluated, its **<who>** clause list is checked. When a **<who>** clause matches the accessor's properties, its **<access>** and **<control>** clauses are evaluated. Access control checking stops at the first match of the **<what>** and **<who>** clause, unless otherwise dictated by the **<control>** clause. Each **<who>** clause list is implicitly terminated by a

```
by * none stop
```

clause that results in stopping the access control with no access privileges granted. Each **<what>** clause list is implicitly terminated by a

```
access to *
        by * none
```

clause that results in granting no access privileges to an otherwise unspecified datum.

THE <WHAT> FIELD

The field **<what>** specifies the entity the access control directive applies to. It can have the forms

dn[.<dnstyle>]=<dnpattern>
filter=<ldapfilter>
attrs=<attrlist>[val[/matchingRule][.<attrstyle>]=<attrval>]

with

<dnstyle>={{exact|base(object)}|regex
|one(level)|sub(tree)|children}
<attrlist>={<attr>|[{!|@}]<objectClass>}[,<attrlist>]
<attrstyle>={{exact|base(object)}|regex
|one(level)|sub(tree)|children}

The statement **dn=<dnpattern>** selects the entries based on their naming context. The **<dnpattern>** is a string representation of the entry's DN. The wildcard ***** stands for all the entries, and it is implied if no **dn** form is given.

The **<dnstyle>** is optional; however, it is recommended to specify it to avoid ambiguities. **Base** (synonym of **baseObject**), the default, or **exact** (an alias of **base**) indicates the entry whose DN is equal to the **<dnpattern>**; **one** (synonym of **onelevel**) indicates all the entries immediately below the **<dnpattern>**, **sub** (synonym of **subtree**) indicates all entries in the subtree at the **<dnpattern>**, **children** indicates all the entries below (subordinate to) the **<dnpattern>**.

If the **<dnstyle>** qualifier is **regex**, then **<dnpattern>** is a POSIX (''extended'') regular expression pattern, as detailed in **regex**(7) and/or **re_format**(7), matching a normalized string representation of the entry's DN. The regex form of the pattern does not (yet) support UTF−8.

The statement **filter=<ldapfilter>** selects the entries based on a valid LDAP filter as described in RFC 4515. A filter of **(objectClass=*)** is implied if no **filter** form is given.

The statement **attrs=<attrlist>** selects the attributes the access control rule applies to. It is a comma-separated list of attribute types, plus the special names **entry**, indicating access to the entry itself, and **children**, indicating access to the entry's children. ObjectClass names may also be specified in this list, which will affect all the attributes that are required and/or allowed by that objectClass. Actually, names in **<attrlist>** that are prefixed by **@** are directly treated as objectClass names. A name prefixed by **!** is also treated as an objectClass, but in this case the access rule affects the attributes that are not required nor allowed by that objectClass. If no **attrs** form is given, **attrs=@extensibleObject** is implied, i.e. all attributes are addressed.

Using the form **attrs=<attr> val[/matchingRule][.<attrstyle>]=<attrval>** specifies access to a particular value of a single attribute. In this case, only a single attribute type may be given. The **<attrstyle>** **exact** (the default) uses the attribute's equality matching rule to compare the value, unless a different (and compatible) matching rule is specified. If the **<attrstyle>** is **regex**, the provided value is used as a POSIX (''extended'') regular expression pattern. If the attribute has DN syntax, the **<attrstyle>** can be any of **base**, **onelevel**, **subtree** or **children**, resulting in base, onelevel, subtree or children match, respectively.

The dn, filter, and attrs statements are additive; they can be used in sequence to select entities the access rule applies to based on naming context, value and attribute type simultaneously.

THE <WHO> FIELD

The field **<who>** indicates whom the access rules apply to. Multiple **<who>** statements can appear in an access control statement, indicating the different access privileges to the same resource that apply to different accessee. It can have the forms

*
anonymous
users
self[.<selfstyle>]

dn[.<dnstyle>[,<modifier>]]=<DN>

dnattr=<attrname>

realanonymous
realusers
realself[.<selfstyle>]

realdn[.<dnstyle>[,<modifier>]]=<DN>
realdnattr=<attrname>

group[/<objectclass>[/<attrname>]]
[.<groupstyle>]=<group>
peername[.<peernamestyle>]=<peername>
sockname[.<style>]=<sockname>
domain[.<domainstyle>[,<modifier>]]=<domain>
sockurl[.<style>]=<sockurl>
set[.<setstyle>]=<pattern>

ssf=<n>
transport_ssf=<n>
tls_ssf=<n>
sasl_ssf=<n>

dynacl/<name>[/<options>][.<dynstyle>][=<pattern>]

with

<style>={exact|regex|expand}
<selfstyle>={level{<n>}}
<dnstyle>={{exact|base(object)}|regex
|one(level)|sub(tree)|children|level{<n>}}
<groupstyle>={exact|expand}
<peernamestyle>={<style>|ip|ipv6|path}
<domainstyle>={exact|regex|sub(tree)}
<setstyle>={exact|regex}
<modifier>={expand}
<name>=aci <pattern>=<attrname>]

They may be specified in combination.

The wildcard ***** refers to everybody.

The keywords prefixed by **real** act as their counterparts without prefix; the checking respectively occurs with the *authentication* DN and the *authorization* DN.

The keyword **anonymous** means access is granted to unauthenticated clients; it is mostly used to limit access to authentication resources (e.g. the **userPassword** attribute) to unauthenticated clients for authentication purposes.

The keyword **users** means access is granted to authenticated clients.

The keyword **self** means access to an entry is allowed to the entry itself (e.g. the entry being accessed and the requesting entry must be the same). It allows the **level{<n>}** style, where <*n*> indicates what ancestor of the DN is to be used in matches. A positive value indicates that the <n>-th ancestor of the user's DN is to be considered; a negative value indicates that the <n>-th ancestor of the target is to be considered. For example, a "*by self.level{1} ...*" clause would match when the object "*dc=example,dc=com*" is accessed by "*cn=User,dc=example,dc=com*". A "*by self.level{-1} ...*" clause would match when the same user accesses the object "*ou=Address Book,cn=User,dc=example,dc=com*".

The statement **dn=<DN>** means that access is granted to the matching DN. The optional style qualifier

dnstyle allows the same choices of the dn form of the **<what>** field. In addition, the **regex** style can exploit substring substitution of submatches in the **<what>** dn.regex clause by using the form **$<digit>**, with **digit** ranging from 0 to 9 (where 0 matches the entire string), or the form **${<digit>+}**, for submatches higher than 9. Since the dollar character is used to indicate a substring replacement, the dollar character that is used to indicate match up to the end of the string must be escaped by a second dollar character, e.g.

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=[^,]+,dc=com$"
   by dn.regex="^uid=$2,dc=[^,]+,dc=com$$" write
```

The style qualifier allows an optional **modifier**. At present, the only type allowed is **expand**, which causes substring substitution of submatches to take place even if **dnstyle** is not **regex**. Note that the **regex** dnstyle in the above example may be of use only if the **<by>** clause needs to be a regex; otherwise, if the value of the second (from the right) **dc=** portion of the DN in the above example were fixed, the form

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
   by dn.exact,expand="uid=$2,dc=example,dc=com" write
```

could be used; if it had to match the value in the **<what>** clause, the form

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=([^,]+),dc=com$"
   by dn.exact,expand="uid=$2,dc=$3,dc=com" write
```

could be used.

Forms of the **<what>** clause other than regex may provide submatches as well. The **base(object)**, the **sub(tree)**, the **one(level)**, and the **children** forms provide **$0** as the match of the entire string. The **sub(tree)**, the **one(level)**, and the **children** forms also provide **$1** as the match of the rightmost part of the DN as defined in the **<what>** clause. This may be useful, for instance, to provide access to all the ancestors of a user by defining

```
access to dn.subtree="dc=com"
   by dn.subtree,expand="$1" read
```

which means that only access to entries that appear in the DN of the **<by>** clause is allowed.

The **level{<n>}** form is an extension and a generalization of the **onelevel** form, which matches all DNs whose <n>-th ancestor is the pattern. So, *level{1}* is equivalent to *onelevel*, and *level{0}* is equivalent to *base*.

It is perfectly useless to give any access privileges to a DN that exactly matches the **rootdn** of the database the ACLs apply to, because it implicitly possesses write privileges for the entire tree of that database. Actually, access control is bypassed for the **rootdn**, to solve the intrinsic chicken-and-egg problem.

The statement **dnattr=<attrname>** means that access is granted to requests whose DN is listed in the entry being accessed under the **<attrname>** attribute.

The statement **group=<group>** means that access is granted to requests whose DN is listed in the group entry whose DN is given by **<group>**. The optional parameters **<objectclass>** and **<attrname>** define the objectClass and the member attributeType of the group entry. The defaults are **groupOfNames** and **member**, respectively. The optional style qualifier **<style>** can be **expand**, which means that **<group>** will be expanded as a replacement string (but not as a regular expression) according to **regex**(7) and/or **re_format**(7), and **exact**, which means that exact match will be used. If the style of the DN portion of the **<what>** clause is regex, the submatches are made available according to **regex**(7) and/or **re_format**(7); other styles provide limited submatches as discussed above about the DN form of the **<by>** clause.

For static groups, the specified attributeType must have **DistinguishedName** or **NameAndOptionalUID** syntax. For dynamic groups the attributeType must be a subtype of the **labeledURI** attributeType. Only LDAP URIs of the form **ldap:///<base>??<scope>?<filter>** will be evaluated in a dynamic group, by searching the local server only.

The statements **peername=<peername>**, **sockname=<sockname>**, **domain=<domain>**, and **sockurl=<sockurl>** mean that the contacting host IP (in the form **IP=<ip>:<port>** for IPv4, or **IP=[<ipv6>]:<port>** for IPv6) or the contacting host named pipe file name (in the form **PATH=<path>** if

connecting through a named pipe) for **peername**, the named pipe file name for **sockname**, the contacting host name for **domain**, and the contacting URL for **sockurl** are compared against **pattern** to determine access. The same **style** rules for pattern match described for the **group** case apply, plus the **regex** style, which implies submatch **expand** and regex match of the corresponding connection parameters. The **exact** style of the **<peername>** clause (the default) implies a case-exact match on the client's **IP**, including the **IP=** prefix and the trailing **:<port>**, or the client's **path**, including the **PATH=** prefix if connecting through a named pipe. The special **ip** style interprets the pattern as **<peername>=<ip>[%<mask>][{<n>}]**, where **<ip>** and **<mask>** are dotted digit representations of the IP and the mask, while **<n>**, delimited by curly brackets, is an optional port. The same applies to IPv6 addresses when the special **ipv6** style is used. When checking access privileges, the IP portion of the **peername** is extracted, eliminating the **IP=** prefix and the **:<port>** part, and it is compared against the **<ip>** portion of the pattern after masking with **<mask>**: *((peername & <mask>) == <ip>)*. As an example, **peername.ip=127.0.0.1** and **peername.ipv6=::1** allow connections only from localhost, **peername.ip=192.168.1.0%255.255.255.0** allows connections from any IP in the 192.168.1 class C domain, and **peername.ip=192.168.1.16%255.255.255.240{9009}** allows connections from any IP in the 192.168.1.[16-31] range of the same domain, only if port 9009 is used. The special **path** style eliminates the **PATH=** prefix from the **peername** when connecting through a named pipe, and performs an exact match on the given pattern. The **<domain>** clause also allows the **subtree** style, which succeeds when a fully qualified name exactly matches the **domain** pattern, or its trailing part, after a **dot**, exactly matches the **domain** pattern. The **expand** style is allowed, implying an **exact** match with submatch expansion; the use of **expand** as a style modifier is considered more appropriate. As an example, **domain.subtree=example.com** will match www.example.com, but will not match www.anotherexample.com. The **domain** of the contacting host is determined by performing a DNS reverse lookup. As this lookup can easily be spoofed, use of the **domain** statement is strongly discouraged. By default, reverse lookups are disabled. The optional **domainstyle** qualifier of the **<domain>** clause allows a **modifier** option; the only value currently supported is **expand**, which causes substring substitution of submatches to take place even if the **domainstyle** is not **regex**, much like the analogous usage in **<dn>** clause.

The statement **set=<pattern>** is undocumented yet.

The statement **dynacl/<name>[/<options>][.<dynstyle>][=<pattern>]** means that access checking is delegated to the admin-defined method indicated by **<name>**, which can be registered at run-time by means of the **moduleload** statement. The fields **<options>**, **<dynstyle>** and **<pattern>** are optional, and are directly passed to the registered parsing routine. Dynacl is experimental; it must be enabled at compile time.

The statement **dynacl/aci[=<attrname>]** means that the access control is determined by the values in the **attrname** of the entry itself. The optional **<attrname>** indicates what attributeType holds the ACI information in the entry. By default, the **OpenLDAPaci** operational attribute is used. ACIs are experimental; they must be enabled at compile time.

The statements **ssf=<n>**, **transport_ssf=<n>**, **tls_ssf=<n>**, and **sasl_ssf=<n>** set the minimum required Security Strength Factor (ssf) needed to grant access. The value should be positive integer.

THE <ACCESS> FIELD

The optional field **<access> ::= [[real]self]{<level>|<priv>}** determines the access level or the specific access privileges the **who** field will have. Its component are defined as

<level> ::= none|disclose|auth|compare|search|read|write|manage
<priv> ::= {=|+|-}{m|w|r|s|c|x|d|0}+

The modifier **self** allows special operations like having a certain access level or privilege only in case the operation involves the name of the user that's requesting the access. It implies the user that requests access is authorized. The modifier **realself** refers to the authenticated DN as opposed to the authorized DN of the **self** modifier. An example is the **selfwrite** access to the member attribute of a group, which allows one to add/delete its own DN from the member list of a group, without affecting other members.

The **level** access model relies on an incremental interpretation of the access privileges. The possible levels are **none**, **disclose**, **auth**, **compare**, **search**, **read**, and **write**. Each access level implies all the preceding ones, thus **manage** grants all access including administrative access,

The **none** access level disallows all access including disclosure on error.

The **disclose** access level allows disclosure of information on error.

The **auth** access level means that one is allowed access to an attribute to perform authentication/authorization operations (e.g. **bind**) with no other access. This is useful to grant unauthenticated clients the least possible access level to critical resources, like passwords.

The **priv** access model relies on the explicit setting of access privileges for each clause. The **=** sign resets previously defined accesses; as a consequence, the final access privileges will be only those defined by the clause. The **+** and **-** signs add/remove access privileges to the existing ones. The privileges are **m** for manage, **w** for write, **r** for read, **s** for search, **c** for compare, **x** for authentication, and **d** for disclose. More than one of the above privileges can be added in one statement. **0** indicates no privileges and is used only by itself (e.g., +0).

If no access is given, it defaults to **+0**.

THE <CONTROL> FIELD

The optional field **<control>** controls the flow of access rule application. It can have the forms

```
stop
continue
break
```

where **stop**, the default, means access checking stops in case of match. The other two forms are used to keep on processing access clauses. In detail, the **continue** form allows for other **<who>** clauses in the same **<access>** clause to be considered, so that they may result in incrementally altering the privileges, while the **break** form allows for other **<access>** clauses that match the same target to be processed. Consider the (silly) example

```
access to dn.subtree="dc=example,dc=com" attrs=cn
        by * =cs break

access to dn.subtree="ou=People,dc=example,dc=com"
        by * +r
```

which allows search and compare privileges to everybody under the "dc=example,dc=com" tree, with the second rule allowing also read in the "ou=People" subtree, or the (even more silly) example

```
access to dn.subtree="dc=example,dc=com" attrs=cn
        by * =cs continue
        by users +r
```

which grants everybody search and compare privileges, and adds read privileges to authenticated clients.

One useful application is to easily grant write privileges to an **updatedn** that is different from the **rootdn**. In this case, since the **updatedn** needs write access to (almost) all data, one can use

```
access to *
        by dn.exact="cn=The Update DN,dc=example,dc=com" write
        by * break
```

as the first access rule. As a consequence, unless the operation is performed with the **updatedn** identity, control is passed straight to the subsequent rules.

OPERATION REQUIREMENTS

Operations require different privileges on different portions of entries. The following summary applies to primary database backends such as the BDB and HDB backends. Requirements for other backends may (and often do) differ.

The **add** operation requires **write (=w)** privileges on the pseudo-attribute **entry** of the entry being added, and **write (=w)** privileges on the pseudo-attribute **children** of the entry's parent. When adding the suffix

entry of a database, write access to **children** of the empty DN ("") is required.

The **bind** operation, when credentials are stored in the directory, requires **auth (=x)** privileges on the attribute the credentials are stored in (usually **userPassword**).

The **compare** operation requires **compare (=c)** privileges on the attribute that is being compared.

The **delete** operation requires **write (=w)** privileges on the pseudo-attribute **entry** of the entry being deleted, and **write (=w)** privileges on the **children** pseudo-attribute of the entry's parent.

The **modify** operation requires **write (=w)** privileges on the attributes being modified.

The **modrdn** operation requires **write (=w)** privileges on the pseudo-attribute **entry** of the entry whose relative DN is being modified, **write (=w)** privileges on the pseudo-attribute **children** of the old and new entry's parents, and **write (=w)** privileges on the attributes that are present in the new relative DN. **Write (=w)** privileges are also required on the attributes that are present in the old relative DN if **deleteoldrdn** is set to 1.

The **search** operation, requires **search (=s)** privileges on the **entry** pseudo-attribute of the searchBase (NOTE: this was introduced with OpenLDAP 2.4). Then, for each entry, it requires **search (=s)** privileges on the attributes that are defined in the filter. The resulting entries are finally tested for **read (=r)** privileges on the pseudo-attribute **entry** (for read access to the entry itself) and for **read (=r)** access on each value of each attribute that is requested. Also, for each **referral** object used in generating continuation references, the operation requires **read (=r)** access on the pseudo-attribute **entry** (for read access to the referral object itself), as well as **read (=r)** access to the attribute holding the referral information (generally the **ref** attribute).

Some internal operations and some **controls** require specific access privileges. The **authzID** mapping and the **proxyAuthz** control require **auth (=x)** privileges on all the attributes that are present in the search filter of the URI regexp maps (the right-hand side of the **authz-regexp** directives). **Auth (=x)** privileges are also required on the **authzTo** attribute of the authorizing identity and/or on the **authzFrom** attribute of the authorized identity. In general, when an internal lookup is performed for authentication or authorization purposes, search-specific privileges (see the access requirements for the search operation illustrated above) are relaxed to **auth**.

Access control to search entries is checked by the frontend, so it is fully honored by all backends; for all other operations and for the discovery phase of the search operation, full ACL semantics is only supported by the primary backends, i.e. **back-bdb**(5), and **back-hdb**(5).

Some other backend, like **back-sql**(5), may fully support them; others may only support a portion of the described semantics, or even differ in some aspects. The relevant details are described in the backend-specific man pages.

CAVEATS

It is strongly recommended to explicitly use the most appropriate **<dnstyle>** in **<what>** and **<who>** clauses, to avoid possible incorrect specifications of the access rules as well as for performance (avoid unnecessary regex matching when an exact match suffices) reasons.

An administrator might create a rule of the form:

```
access to dn.regex="dc=example,dc=com"
        by ...
```

expecting it to match all entries in the subtree "dc=example,dc=com". However, this rule actually matches any DN which contains anywhere the substring "dc=example,dc=com". That is, the rule matches both "uid=joe,dc=example,dc=com" and "dc=example,dc=com,uid=joe".

To match the desired subtree, the rule would be more precisely written:

```
access to dn.regex="^(.+,)?dc=example,dc=com$"
        by ...
```

For performance reasons, it would be better to use the subtree style.

```
access to dn.subtree="dc=example,dc=com"
        by ...
```

When writing submatch rules, it may be convenient to avoid unnecessary **regex <dnstyle>** use; for instance, to allow access to the subtree of the user that matches the **<what>** clause, one could use

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
        by dn.regex="^uid=$2,dc=example,dc=com$$" write
        by ...
```

However, since all that is required in the **<by>** clause is substring expansion, a more efficient solution is

```
access to dn.regex="^(.+,)?uid=([^,]+),dc=example,dc=com$"
        by dn.exact,expand="uid=$2,dc=example,dc=com" write
        by ...
```

In fact, while a **<dnstyle>** of **regex** implies substring expansion, **exact**, as well as all the other DN specific **<dnstyle>** values, does not, so it must be explicitly requested.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd(8), **slapd-***(5), **slapacl**(8), **regex**(7), **re_format**(7)

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapd.backends − backends for slapd, the stand-alone LDAP daemon

DESCRIPTION

The **slapd**(8) daemon can use a variety of different backends for serving LDAP requests. Backends may be compiled statically into slapd, or when module support is enabled, they may be dynamically loaded. Multiple instances of a backend can be configured, to serve separate databases from the same slapd server.

Configuration options for each backend are documented separately in the corresponding **slapd-<backend>**(5) manual pages.

bdb This is the recommended primary backend for a normal slapd database. It takes care to configure it properly. It uses the transactional database interface of the Oracle Berkeley DB (BDB) package to store data.

config This backend is used to manage the configuration of slapd at run-time. Unlike other backends, only a single instance of the **config** backend may be defined. It also instantiates itself automatically, so it is always present even if not explicitly defined in the **slapd.conf**(5) file.

dnssrv This backend is experimental. It serves up referrals based upon SRV resource records held in the Domain Name System.

hdb This is a variant of the **bdb** backend that uses a hierarchical database layout. This layout stores entry DNs more efficiently than the **bdb** backend, using less space and requiring less work to create, delete, and rename entries. It is also one of the few backends to support subtree renames.

ldap This backend acts as a proxy to forward incoming requests to another LDAP server.

ldif This database uses the filesystem to build the tree structure of the database, using plain ascii files to store data. Its usage should be limited to very simple databases, where performance is not a requirement. This backend also supports subtree renames.

meta This backend performs basic LDAP proxying with respect to a set of remote LDAP servers. It is an enhancement of the **ldap** backend.

monitor
This backend provides information about the running status of the slapd daemon. Only a single instance of the **monitor** backend may be defined.

null Operations in this backend succeed but do nothing.

passwd
This backend is provided for demonstration purposes only. It serves up user account information from the system **passwd**(5) file.

perl This backend embeds a **perl**(1) interpreter into slapd. It runs Perl subroutines to implement LDAP operations.

relay This backend is experimental. It redirects LDAP operations to another database in the same server, based on the naming context of the request. Its use requires the **rwm** overlay (see **slapo-rwm**(5) for details) to rewrite the naming context of the request. It is primarily intended to implement virtual views on databases that actually store data.

shell This backend executes external programs to implement LDAP operations. It is primarily intended to be used in prototypes.

sql This backend is experimental. It services LDAP requests from an SQL database.

FILES

ETCDIR/slapd.conf
default slapd configuration file

ETCDIR/slapd.d
: default slapd configuration directory

SEE ALSO

ldap(3), **slapd−bdb**(5), **slapd−config**(5), **slapd−dnssrv**(5), **slapd−hdb**(5), **slapd−ldap**(5), **slapd−ldif**(5), **slapd−meta**(5), **slapd−monitor**(5), **slapd−null**(5), **slapd−passwd**(5), **slapd−perl**(5), **slapd−relay**(5), **slapd−shell**(5), **slapd−sql**(5), **slapd.conf**(5), **slapd.overlays**(5), **slapd**(8). "OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

NAME

slapd.backends − backends for slapd, the stand-alone LDAP daemon

DESCRIPTION

The **slapd**(8) daemon can use a variety of different backends for serving LDAP requests. Backends may be compiled statically into slapd, or when module support is enabled, they may be dynamically loaded. Multiple instances of a backend can be configured, to serve separate databases from the same slapd server.

Configuration options for each backend are documented separately in the corresponding **slapd-<backend>**(5) manual pages.

bdb This is the recommended primary backend for a normal slapd database. It takes care to configure it properly. It uses the transactional database interface of the Oracle Berkeley DB (BDB) package to store data.

config This backend is used to manage the configuration of slapd at run-time. Unlike other backends, only a single instance of the **config** backend may be defined. It also instantiates itself automatically, so it is always present even if not explicitly defined in the **slapd.conf**(5) file.

dnssrv This backend is experimental. It serves up referrals based upon SRV resource records held in the Domain Name System.

hdb This is a variant of the **bdb** backend that uses a hierarchical database layout. This layout stores entry DNs more efficiently than the **bdb** backend, using less space and requiring less work to create, delete, and rename entries. It is also one of the few backends to support subtree renames.

ldap This backend acts as a proxy to forward incoming requests to another LDAP server.

ldif This database uses the filesystem to build the tree structure of the database, using plain ascii files to store data. Its usage should be limited to very simple databases, where performance is not a requirement. This backend also supports subtree renames.

meta This backend performs basic LDAP proxying with respect to a set of remote LDAP servers. It is an enhancement of the **ldap** backend.

monitor
This backend provides information about the running status of the slapd daemon. Only a single instance of the **monitor** backend may be defined.

null Operations in this backend succeed but do nothing.

passwd
This backend is provided for demonstration purposes only. It serves up user account information from the system **passwd**(5) file.

perl This backend embeds a **perl**(1) interpreter into slapd. It runs Perl subroutines to implement LDAP operations.

relay This backend is experimental. It redirects LDAP operations to another database in the same server, based on the naming context of the request. Its use requires the **rwm** overlay (see **slapo-rwm**(5) for details) to rewrite the naming context of the request. It is primarily intended to implement virtual views on databases that actually store data.

shell This backend executes external programs to implement LDAP operations. It is primarily intended to be used in prototypes.

sql This backend is experimental. It services LDAP requests from an SQL database.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

/etc/openldap/slapd.d
default slapd configuration directory

SEE ALSO

ldap(3), **slapd−bdb**(5), **slapd−config**(5), **slapd−dnssrv**(5), **slapd−hdb**(5), **slapd−ldap**(5), **slapd−ldif**(5), **slapd−meta**(5), **slapd−monitor**(5), **slapd−null**(5), **slapd−passwd**(5), **slapd−perl**(5), **slapd−relay**(5), **slapd−shell**(5), **slapd−sql**(5), **slapd.conf**(5), **slapd.overlays**(5), **slapd**(8). "OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapd.conf − configuration file for slapd, the stand-alone LDAP daemon

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The file **ETCDIR/slapd.conf** contains configuration information for the **slapd**(8) daemon. This configuration file is also used by the SLAPD tools **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), and **slaptest**(8).

The **slapd.conf** file consists of a series of global configuration options that apply to **slapd** as a whole (including all backends), followed by zero or more database backend definitions that contain information specific to a backend instance. The configuration options are case-insensitive; their value, on a case by case basis, may be case-sensitive.

The general format of **slapd.conf** is as follows:

```
# comment - these options apply to every database
<global configuration options>
# first database definition & configuration options
database <backend 1 type>
<configuration options specific to backend 1>
# subsequent database definitions & configuration options
...
```

As many backend-specific sections as desired may be included. Global options can be overridden in a backend (for options that appear more than once, the last appearance in the **slapd.conf** file is used).

If a line begins with white space, it is considered a continuation of the previous line. No physical line should be over 2000 bytes long.

Blank lines and comment lines beginning with a '#' character are ignored. Note: continuation lines are unwrapped before comment processing is applied.

Arguments on configuration lines are separated by white space. If an argument contains white space, the argument should be enclosed in double quotes. If an argument contains a double quote ('"') or a backslash character ('\'), the character should be preceded by a backslash character.

The specific configuration options available are discussed below in the Global Configuration Options, General Backend Options, and General Database Options. Backend-specific options are discussed in the **slapd-<backend>(5)** manual pages. Refer to the "OpenLDAP Administrator's Guide" for more details on the slapd configuration file.

GLOBAL CONFIGURATION OPTIONS

Options described in this section apply to all backends, unless specifically overridden in a backend definition. Arguments that should be replaced by actual text are shown in brackets <>.

access to <what> [by <who> <access> <control>]+

Grant access (specified by <access>) to a set of entries and/or attributes (specified by <what>) by one or more requestors (specified by <who>). If no access controls are present, the default policy allows anyone and everyone to read anything but restricts updates to rootdn. (e.g., "access to * by * read"). The rootdn can always read and write EVERYTHING! See **slapd.access**(5) and the "OpenLDAP's Administrator's Guide" for details.

allow <features>

Specify a set of features (separated by white space) to allow (default none). **bind_v2** allows acceptance of LDAPv2 bind requests. Note that **slapd**(8) does not truly implement LDAPv2 (RFC 1777), now Historic (RFC 3494). **bind_anon_cred** allows anonymous bind when credentials are not empty (e.g. when DN is empty). **bind_anon_dn** allows unauthenticated (anonymous) bind when DN is not empty. **update_anon** allows unauthenticated (anonymous) update operations to be processed (subject to access controls and other administrative limits). **proxy_authz_anon**

allows unauthenticated (anonymous) proxy authorization control to be processed (subject to access controls, authorization and other administrative limits).

argsfile <filename>
The (absolute) name of a file that will hold the **slapd** server's command line options if started without the debugging command line option.

attributeoptions [option-name]...
Define tagging attribute options or option tag/range prefixes. Options must not end with '-', prefixes must end with '-'. The 'lang-' prefix is predefined. If you use the **attributeoptions** directive, 'lang-' will no longer be defined and you must specify it explicitly if you want it defined.

An attribute description with a tagging option is a subtype of that attribute description without the option. Except for that, options defined this way have no special semantics. Prefixes defined this way work like the 'lang-' options: They define a prefix for tagging options starting with the prefix. That is, if you define the prefix 'x-foo-', you can use the option 'x-foo-bar'. Furthermore, in a search or compare, a prefix or range name (with a trailing '-') matches all options starting with that name, as well as the option with the range name sans the trailing '-'. That is, 'x-foo-bar-' matches 'x-foo-bar' and 'x-foo-bar-baz'.

RFC 4520 reserves options beginning with 'x-' for private experiments. Other options should be registered with IANA, see RFC 4520 section 3.5. OpenLDAP also has the 'binary' option built in, but this is a transfer option, not a tagging option.

attributetype (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oid>] [EQUALITY <oid>] [ORDERING <oid>] [SUBSTR <oid>] [SYNTAX <oidlen>] [SINGLE−VALUE] [COLLECTIVE] [NO−USER−MODIFICATION] [USAGE <attributeUsage>])
Specify an attribute type using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **objectidentifier** description.)

authz-policy <policy>
Used to specify which rules to use for Proxy Authorization. Proxy authorization allows a client to authenticate to the server using one user's credentials, but specify a different identity to use for authorization and access control purposes. It essentially allows user A to login as user B, using user A's password. The **none** flag disables proxy authorization. This is the default setting. The **from** flag will use rules in the *authzFrom* attribute of the authorization DN. The **to** flag will use rules in the *authzTo* attribute of the authentication DN. The **any** flag, an alias for the deprecated value of **both**, will allow any of the above, whatever succeeds first (checked in **to**, **from** sequence. The **all** flag requires both authorizations to succeed.

The rules are mechanisms to specify which identities are allowed to perform proxy authorization. The *authzFrom* attribute in an entry specifies which other users are allowed to proxy login to this entry. The *authzTo* attribute in an entry specifies which other users this user can authorize as. Use of *authzTo* rules can be easily abused if users are allowed to write arbitrary values to this attribute. In general the *authzTo* attribute must be protected with ACLs such that only privileged users can modify it. The value of *authzFrom* and *authzTo* describes an **identity** or a set of identities; it can take five forms:

ldap:///<base>??[<scope>]?<filter>
dn[.<dnstyle>]:<pattern>
u[<mech>[<realm>]]:<pattern>
group[/objectClass[/attributeType]]:<pattern>
<pattern>

<dnstyle>:={exact|onelevel|children|subtree|regex}

The first form is a valid LDAP **URI** where the *<host>:<port>*, the *<attrs>* and the *<extensions>* portions must be absent, so that the search occurs locally on either *authzFrom* or *authzTo*. The second form is a **DN**, with the optional style modifiers *exact*, *onelevel*, *children*, and *subtree* for exact, onelevel, children and subtree matches, which cause *<pattern>* to be normalized according to the DN normalization rules, or the special *regex* style, which causes the *<pattern>* to be treated as a POSIX (''extended'') regular expression, as discussed in **regex**(7) and/or **re_format**(7). A pattern of * means any non-anonymous DN. The third form is a SASL **id**, with the optional fields *<mech>* and *<realm>* that allow to specify a SASL **mechanism**, and eventually a SASL **realm**, for those mechanisms that support one. The need to allow the specification of a mechanism is still debated, and users are strongly discouraged to rely on this possibility. The fourth form is a group specification, consisting of the keyword **group**, optionally followed by the specification of the group **objectClass** and member **attributeType**. The group with DN **<pattern>** is searched with base scope, and in case of match, the values of the member **attributeType** are searched for the asserted DN. For backwards compatibility, if no identity type is provided, i.e. only **<pattern>** is present, an *exact DN* is assumed; as a consequence, **<pattern>** is subjected to DN normalization. Since the interpretation of *authzFrom* and *authzTo* can impact security, users are strongly encouraged to explicitly set the type of identity specification that is being used. A subset of these rules can be used as third arg in the **authz-regexp** statement (see below); significantly, the *URI* and the *dn.exact:<dn>* forms.

authz-regexp <match> <replace>

Used by the authentication framework to convert simple user names, such as provided by SASL subsystem, to an LDAP DN used for authorization purposes. Note that the resultant DN need not refer to an existing entry to be considered valid. When an authorization request is received from the SASL subsystem, the SASL **USERNAME**, **REALM**, and **MECHANISM** are taken, when available, and combined into a name of the form

UID=<username>[[,CN=<realm>],CN=<mechanism>],CN=auth

This name is then compared against the **match** POSIX (''extended'') regular expression, and if the match is successful, the name is replaced with the **replace** string. If there are wildcard strings in the **match** regular expression that are enclosed in parenthesis, e.g.

UID=([^,]*),CN=.*

then the portion of the name that matched the wildcard will be stored in the numbered placeholder variable $1. If there are other wildcard strings in parenthesis, the matching strings will be in $2, $3, etc. up to $9. The placeholders can then be used in the **replace** string, e.g.

UID=$1,OU=Accounts,DC=example,DC=com

The replaced name can be either a DN, i.e. a string prefixed by "dn:", or an LDAP URI. If the latter, the server will use the URI to search its own database(s) and, if the search returns exactly one entry, the name is replaced by the DN of that entry. The LDAP URI must have no hostport, attrs, or extensions components, but the filter is mandatory, e.g.

ldap:///OU=Accounts,DC=example,DC=com??one?(UID=$1)

The protocol portion of the URI must be strictly **ldap**. Note that this search is subject to access controls. Specifically, the authentication identity must have "auth" access in the subject.

Multiple **authz-regexp** options can be given in the configuration file to allow for multiple matching and replacement patterns. The matching patterns are checked in the order they appear in the file, stopping at the first successful match.

concurrency <integer>
Specify a desired level of concurrency. Provided to the underlying thread system as a hint. The default is not to provide any hint.

conn_max_pending <integer>
Specify the maximum number of pending requests for an anonymous session. If requests are submitted faster than the server can process them, they will be queued up to this limit. If the limit is exceeded, the session is closed. The default is 100.

conn_max_pending_auth <integer>
Specify the maximum number of pending requests for an authenticated session. The default is 1000.

defaultsearchbase <dn>
Specify a default search base to use when client submits a non-base search request with an empty base DN. Base scoped search requests with an empty base DN are not affected.

disallow <features>
Specify a set of features (separated by white space) to disallow (default none). **bind_anon** disables acceptance of anonymous bind requests. Note that this setting does not prohibit anonymous directory access (See "require authc"). **bind_simple** disables simple (bind) authentication. **tls_2_anon** disables forcing session to anonymous status (see also **tls_authc**) upon StartTLS operation receipt. **tls_authc** disallows the StartTLS operation if authenticated (see also **tls_2_anon**).

ditcontentrule (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [AUX <oids>] [MUST <oids>] [MAY <oids>] [NOT <oids>])
Specify an DIT Content Rule using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **objectidentifier** description.)

gentlehup { on | off }
A SIGHUP signal will only cause a 'gentle' shutdown-attempt: **Slapd** will stop listening for new connections, but will not close the connections to the current clients. Future write operations return unwilling-to-perform, though. Slapd terminates when all clients have closed their connections (if they ever do), or − as before − if it receives a SIGTERM signal. This can be useful if you wish to terminate the server and start a new **slapd** server **with another database,** without disrupting the currently active clients. The default is off. You may wish to use **idletimeout** along with this option.

idletimeout <integer>
Specify the number of seconds to wait before forcibly closing an idle client connection. A idletimeout of 0 disables this feature. The default is 0.

include <filename>
Read additional configuration information from the given file before continuing with the next line of the current file.

index_intlen <integer>
Specify the key length for ordered integer indices. The most significant bytes of the binary integer will be used for index keys. The default value is 4, which provides exact indexing for 31 bit values. A floating point representation is used to index too large values.

index_substr_if_minlen <integer>
Specify the minimum length for subinitial and subfinal indices. An attribute value must have at least this many characters in order to be processed by the indexing functions. The default is 2.

index_substr_if_maxlen <integer>
Specify the maximum length for subinitial and subfinal indices. Only this many characters of an attribute value will be processed by the indexing functions; any excess characters are ignored. The default is 4.

index_substr_any_len <integer>
Specify the length used for subany indices. An attribute value must have at least this many characters in order to be processed. Attribute values longer than this length will be processed in segments of this length. The default is 4. The subany index will also be used in subinitial and subfinal index lookups when the filter string is longer than the *index_substr_if_maxlen* value.

index_substr_any_step <integer>
Specify the steps used in subany index lookups. This value sets the offset for the segments of a filter string that are processed for a subany index lookup. The default is 2. For example, with the default values, a search using this filter "cn=*abcdefgh*" would generate index lookups for "abcd", "cdef", and "efgh".

Note: Indexing support depends on the particular backend in use. Also, changing these settings will generally require deleting any indices that depend on these parameters and recreating them with **slapindex**(8).

localSSF <SSF>
Specifies the Security Strength Factor (SSF) to be given local LDAP sessions, such as those to the ldapi:// listener. For a description of SSF values, see **sasl-secprops**'s **minssf** option description. The default is 71.

logfile <filename>
Specify a file for recording debug log messages. By default these messages only go to stderr and are not recorded anywhere else. Specifying a logfile copies messages to both stderr and the logfile.

loglevel <integer> [...]
Specify the level at which debugging statements and operation statistics should be syslogged (currently logged to the **syslogd**(8) LOG_LOCAL4 facility). They must be considered subsystems rather than increasingly verbose log levels. Some messages with higher priority are logged regardless of the configured loglevel as soon as any logging is configured. Log levels are additive, and available levels are:

1	**(0x1 trace)** trace function calls
2	**(0x2 packets)** debug packet handling
4	**(0x4 args)** heavy trace debugging (function args)
8	**(0x8 conns)** connection management
16	**(0x10 BER)** print out packets sent and received
32	**(0x20 filter)** search filter processing
64	**(0x40 config)** configuration file processing
128	**(0x80 ACL)** access control list processing
256	**(0x100 stats)** connections, LDAP operations, results (recommended)
512	**(0x200 stats2)** stats log entries sent
1024	**(0x400 shell)** print communication with shell backends
2048	**(0x800 parse)** entry parsing
16384	**(0x4000 sync)** LDAPSync replication
32768	**(0x8000 none)** only messages that get logged whatever log level is set

The desired log level can be input as a single integer that combines the (ORed) desired levels, both in decimal or in hexadecimal notation, as a list of integers (that are ORed internally), or as a list of

the names that are shown between brackets, such that

loglevel 129
loglevel 0x81
loglevel 128 1
loglevel 0x80 0x1
loglevel acl trace

are equivalent. The keyword **any** can be used as a shortcut to enable logging at all levels (equivalent to -1). The keyword **none**, or the equivalent integer representation, causes those messages that are logged regardless of the configured loglevel to be logged. In fact, if loglevel is set to 0, no logging occurs, so at least the **none** level is required to have high priority messages logged.

The loglevel defaults to **stats**. This level should usually also be included when using other loglevels, to help analyze the logs.

moduleload <filename>
Specify the name of a dynamically loadable module to load. The filename may be an absolute path name or a simple filename. Non-absolute names are searched for in the directories specified by the **modulepath** option. This option and the **modulepath** option are only usable if slapd was compiled with --enable-modules.

modulepath <pathspec>
Specify a list of directories to search for loadable modules. Typically the path is colon-separated but this depends on the operating system.

objectclass (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oids>] [{ ABSTRACT | STRUCTURAL | AUXILIARY }] [MUST <oids>] [MAY <oids>])
Specify an objectclass using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the object class OID. (See the **objectidentifier** description.) Object classes are "STRUCTURAL" by default.

objectidentifier <name> { <oid> | <name>[:<suffix>] }
Define a string name that equates to the given OID. The string can be used in place of the numeric OID in objectclass and attribute definitions. The name can also be used with a suffix of the form ":xx" in which case the value "oid.xx" will be used.

password-hash <hash> [<hash>...]
This option configures one or more hashes to be used in generation of user passwords stored in the userPassword attribute during processing of LDAP Password Modify Extended Operations (RFC 3062). The <hash> must be one of **{SSHA}**, **{SHA}**, **{SMD5}**, **{MD5}**, **{CRYPT}**, and **{CLEARTEXT}**. The default is **{SSHA}**.

{SHA} and **{SSHA}** use the SHA-1 algorithm (FIPS 160-1), the latter with a seed.

{MD5} and **{SMD5}** use the MD5 algorithm (RFC 1321), the latter with a seed.

{CRYPT} uses the **crypt**(3).

{CLEARTEXT} indicates that the new password should be added to userPassword as clear text.

Note that this option does not alter the normal user applications handling of userPassword during LDAP Add, Modify, or other LDAP operations.

password−crypt−salt−format <format>
Specify the format of the salt passed to **crypt**(3) when generating {CRYPT} passwords (see **password−hash**) during processing of LDAP Password Modify Extended Operations (RFC 3062).

This string needs to be in **sprintf**(3) format and may include one (and only one) %s conversion. This conversion will be substituted with a string of random characters from [A−Za−z0−9./]. For example, "%.2s" provides a two character salt and "1%.8s" tells some versions of crypt(3) to use an MD5 algorithm and provides 8 random characters of salt. The default is "%s", which provides 31 characters of salt.

pidfile <filename>

The (absolute) name of a file that will hold the **slapd** server's process ID (see **getpid**(2)) if started without the debugging command line option.

referral <url>

Specify the referral to pass back when **slapd**(8) cannot find a local database to handle a request. If specified multiple times, each url is provided.

require <conditions>

Specify a set of conditions (separated by white space) to require (default none). The directive may be specified globally and/or per-database; databases inherit global conditions, so per-database specifications are additive. **bind** requires bind operation prior to directory operations. **LDAPv3** requires session to be using LDAP version 3. **authc** requires authentication prior to directory operations. **SASL** requires SASL authentication prior to directory operations. **strong** requires strong authentication prior to directory operations. The strong keyword allows protected "simple" authentication as well as SASL authentication. **none** may be used to require no conditions (useful to clear out globally set conditions within a particular database); it must occur first in the list of conditions.

reverse-lookup on | off

Enable/disable client name unverified reverse lookup (default is **off** if compiled with --enable-rlookups).

rootDSE <file>

Specify the name of an LDIF(5) file containing user defined attributes for the root DSE. These attributes are returned in addition to the attributes normally produced by slapd.

The root DSE is an entry with information about the server and its capabilities, in operational attributes. It has the empty DN, and can be read with e.g.:

```
ldapsearch -x -b "" -s base "+"
```

See RFC 4512 section 5.1 for details.

sasl-host <fqdn>

Used to specify the fully qualified domain name used for SASL processing.

sasl-realm <realm>

Specify SASL realm. Default is empty.

sasl-secprops <properties>

Used to specify Cyrus SASL security properties. The **none** flag (without any other properties) causes the flag properties default, "noanonymous,noplain", to be cleared. The **noplain** flag disables mechanisms susceptible to simple passive attacks. The **noactive** flag disables mechanisms susceptible to active attacks. The **nodict** flag disables mechanisms susceptible to passive dictionary attacks. The **noanonymous** flag disables mechanisms which support anonymous login. The **forwardsec** flag require forward secrecy between sessions. The **passcred** require mechanisms which pass client credentials (and allow mechanisms which can pass credentials to do so). The **minssf=<factor>** property specifies the minimum acceptable *security strength factor* as an integer approximate to effective key length used for encryption. 0 (zero) implies no protection, 1 implies integrity protection only, 56 allows DES or other weak ciphers, 112 allows triple DES and other strong ciphers, 128 allows RC4, Blowfish and other modern strong ciphers. The default is 0. The **maxssf=<factor>** property specifies the maximum acceptable *security strength factor* as an integer (see minssf description). The default is INT_MAX. The **maxbufsize=<size>** property specifies the maximum security layer receive

buffer size allowed. 0 disables security layers. The default is 65536.

schemadn <dn>
Specify the distinguished name for the subschema subentry that controls the entries on this server. The default is "cn=Subschema".

security <factors>
Specify a set of security strength factors (separated by white space) to require (see **sasl-secprops**'s **minssf** option for a description of security strength factors). The directive may be specified globally and/or per-database. **ssf=<n>** specifies the overall security strength factor. **transport=<n>** specifies the transport security strength factor. **tls=<n>** specifies the TLS security strength factor. **sasl=<n>** specifies the SASL security strength factor. **update_ssf=<n>** specifies the overall security strength factor to require for directory updates. **update_transport=<n>** specifies the transport security strength factor to require for directory updates. **update_tls=<n>** specifies the TLS security strength factor to require for directory updates. **update_sasl=<n>** specifies the SASL security strength factor to require for directory updates. **simple_bind=<n>** specifies the security strength factor required for *simple* username/password authentication. Note that the **transport** factor is measure of security provided by the underlying transport, e.g. ldapi:// (and eventually IPSEC). It is not normally used.

serverID <integer> [<URL>]
Specify an integer ID from 0 to 4095 for this server (limited to 3 hexadecimal digits). These IDs are required when using multimaster replication and each master must have a unique ID. If the URL is provided, this directive may be specified multiple times, providing a complete list of participating servers and their IDs. The fully qualified hostname of each server should be used in the supplied URLs. The IDs are used in the "replica id" field of all CSNs generated by the specified server. The default value is zero. Example:

serverID 1

sizelimit {<integer>|unlimited}

sizelimit size[.{soft|hard|unchecked}]=<integer> [...]
Specify the maximum number of entries to return from a search operation. The default size limit is 500. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the size limits. Extra args can be added on the same line. See **limits** for an explanation of the different flags.

sockbuf_max_incoming <integer>
Specify the maximum incoming LDAP PDU size for anonymous sessions. The default is 262143.

sockbuf_max_incoming_auth <integer>
Specify the maximum incoming LDAP PDU size for authenticated sessions. The default is 4194303.

sortvals <attr> [...]
Specify a list of multi-valued attributes whose values will always be maintained in sorted order. Using this option will allow Modify, Compare, and filter evaluations on these attributes to be performed more efficiently. The resulting sort order depends on the attributes' syntax and matching rules and may not correspond to lexical order or any other recognizable order.

threads <integer>
Specify the maximum size of the primary thread pool. The default is 16; the minimum value is 2.

timelimit {<integer>|unlimited}

timelimit time[.{soft|hard}]=<integer> [...]
Specify the maximum number of seconds (in real time) **slapd** will spend answering a search request. The default time limit is 3600. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the time limits. Extra args can be added on the same line. See **limits** for an explanation of the different flags.

tool-threads <integer>
Specify the maximum number of threads to use in tool mode. This should not be greater than the number of CPUs in the system. The default is 1.

TLS OPTIONS

If **slapd** is built with support for Transport Layer Security, there are more options you can specify.

TLSCipherSuite <cipher-suite-spec>
Permits configuring what ciphers will be accepted and the preference order. <cipher-suite-spec> should be a cipher specification for OpenSSL. Example:

TLSCipherSuite HIGH:MEDIUM:+SSLv2

To check what ciphers a given spec selects, use:

openssl ciphers -v <cipher-suite-spec>

To obtain the list of ciphers in GNUtls use:

gnutls-cli -l

TLSCACertificateFile <filename>
Specifies the file that contains certificates for all of the Certificate Authorities that **slapd** will recognize.

TLSCACertificatePath <path>
Specifies the path of a directory that contains Certificate Authority certificates in separate individual files. Usually only one of this or the TLSCACertificateFile is used. This directive is not supported when using GNUtls.

TLSCertificateFile <filename>
Specifies the file that contains the **slapd** server certificate.

TLSCertificateKeyFile <filename>
Specifies the file that contains the **slapd** server private key that matches the certificate stored in the **TLSCertificateFile** file. Currently, the private key must not be protected with a password, so it is of critical importance that it is protected carefully.

TLSDHParamFile <filename>
This directive specifies the file that contains parameters for Diffie-Hellman ephemeral key exchange. This is required in order to use a DSA certificate on the server. If multiple sets of parameters are present in the file, all of them will be processed. Note that setting this option may also enable Anonymous Diffie-Hellman key exchanges in certain non-default cipher suites. You should append "!ADH" to your cipher suites if you have changed them from the default, otherwise no certificate exchanges or verification will be done. When using GNUtls these parameters are always generated randomly so this directive is ignored.

TLSRandFile <filename>
Specifies the file to obtain random bits from when /dev/[u]random is not available. Generally set to the name of the EGD/PRNGD socket. The environment variable RANDFILE can also be used to specify the filename. This directive is ignored with GNUtls.

TLSVerifyClient <level>
Specifies what checks to perform on client certificates in an incoming TLS session, if any. The **<level>** can be specified as one of the following keywords:

never This is the default. **slapd** will not ask the client for a certificate.

allow The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, it will be ignored and the session proceeds

normally.

try The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, the session is immediately terminated.

demand | hard | true
These keywords are all equivalent, for compatibility reasons. The client certificate is requested. If no certificate is provided, or a bad certificate is provided, the session is immediately terminated.

Note that a valid client certificate is required in order to use the SASL EXTERNAL authentication mechanism with a TLS session. As such, a non-default **TLSVerifyClient** setting must be chosen to enable SASL EXTERNAL authentication.

TLSCRLCheck <level>
Specifies if the Certificate Revocation List (CRL) of the CA should be used to verify if the client certificates have not been revoked. This requires **TLSCACertificatePath** parameter to be set. This directive is ignored with GNUtls. **<level>** can be specified as one of the following keywords:

none No CRL checks are performed

peer Check the CRL of the peer certificate

all Check the CRL for a whole certificate chain

TLSCRLFile <filename>
Specifies a file containing a Certificate Revocation List to be used for verifying that certificates have not been revoked. This directive is only valid when using GNUtls.

GENERAL BACKEND OPTIONS

Options in this section only apply to the configuration file section for the specified backend. They are supported by every type of backend.

backend <databasetype>
Mark the beginning of a backend definition. <databasetype> should be one of **bdb**, **config**, **dnssrv**, **hdb**, **ldap**, **ldif**, **meta**, **monitor**, **null**, **passwd**, **perl**, **relay**, **shell**, or **sql**, depending on which backend will serve the database.

GENERAL DATABASE OPTIONS

Options in this section only apply to the configuration file section for the database in which they are defined. They are supported by every type of backend. Note that the **database** and at least one **suffix** option are mandatory for each database.

database <databasetype>
Mark the beginning of a new database instance definition. <databasetype> should be one of **bdb**, **config**, **dnssrv**, **hdb**, **ldap**, **ldif**, **meta**, **monitor**, **null**, **passwd**, **perl**, **relay**, **shell**, or **sql**, depending on which backend will serve the database.

hidden on | off
Controls whether the database will be used to answer queries. A database that is hidden will never be selected to answer any queries, and any suffix configured on the database will be ignored in checks for conflicts with other databases. By default, hidden is off.

lastmod on | off
Controls whether **slapd** will automatically maintain the modifiersName, modifyTimestamp, creatorsName, and createTimestamp attributes for entries. It also controls the entryCSN and entryUUID attributes, which are needed by the syncrepl provider. By default, lastmod is on.

limits <who> <limit> [<limit> [...]]
Specify time and size limits based on who initiated an operation. The argument **who** can be any of

anonymous | users | [dn[.<style>]=]<pattern> | group[/oc[/at]]=<pattern>

with

<style> ::= exact | base | onelevel | subtree | children | regex | anonymous

The term **anonymous** matches all unauthenticated clients. The term **users** matches all authenticated clients; otherwise an **exact** dn pattern is assumed unless otherwise specified by qualifying the (optional) key string **dn** with **exact** or **base** (which are synonyms), to require an exact match; with **onelevel**, to require exactly one level of depth match; with **subtree**, to allow any level of depth match, including the exact match; with **children**, to allow any level of depth match, not including the exact match; **regex** explicitly requires the (default) match based on POSIX (''extended'') regular expression pattern. Finally, **anonymous** matches unbound operations; the **pattern** field is ignored. The same behavior is obtained by using the **anonymous** form of the **who** clause. The term **group**, with the optional objectClass **oc** and attributeType **at** fields, followed by **pattern**, sets the limits for any DN listed in the values of the **at** attribute (default **member**) of the **oc** group objectClass (default **groupOfNames**) whose DN exactly matches **pattern**.

The currently supported limits are **size** and **time**.

The syntax for time limits is **time[.{soft|hard}]=<integer>**, where *integer* is the number of seconds slapd will spend answering a search request. If no time limit is explicitly requested by the client, the **soft** limit is used; if the requested time limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for time limits smaller or equal to the **hard** limit are honored. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

The syntax for size limits is **size[.{soft|hard|unchecked}]=<integer>**, where *integer* is the maximum number of entries slapd will return answering a search request. If no size limit is explicitly requested by the client, the **soft** limit is used; if the requested size limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for size limits smaller or equal to the **hard** limit are honored. The **unchecked** specifier sets a limit on the number of candidates a search request is allowed to examine. The rationale behind it is that searches for non-properly indexed attributes may result in large sets of candidates, which must be examined by **slapd**(8) to determine whether they match the search filter or not. The **unchecked** limit provides a means to drop such operations before they are even started. If the selected candidates exceed the **unchecked** limit, the search will abort with *Unwilling to perform*. If it is set to the keyword *unlimited*, no limit is applied (the default). If it is set to *disable*, the search is not even performed; this can be used to disallow searches for a specific set of users. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

In case of no match, the global limits are used. The default values are the same of **sizelimit** and **timelimit**; no limit is set on **unchecked**.

If **pagedResults** control is requested, the **hard** size limit is used by default, because the request of a specific page size is considered an explicit request for a limitation on the number of entries to be returned. However, the size limit applies to the total count of entries returned within the search, and not to a single page. Additional size limits may be enforced; the syntax is **size.pr={<integer>|noEstimate|unlimited}**, where *integer* is the max page size if no explicit limit is set; the keyword *noEstimate* inhibits the server from returning an estimate of the total number of entries that might be returned (note: the current implementation does not return any estimate). The keyword *unlimited* indicates that no limit is applied to the pagedResults control page size. The

syntax **size.prtotal={<integer>|unlimited|disabled}** allows to set a limit on the total number of entries that a pagedResults control allows to return. By default it is set to the **hard** limit. When set, *integer* is the max number of entries that the whole search with pagedResults control can return. Use *unlimited* to allow unlimited number of entries to be returned, e.g. to allow the use of the pagedResults control as a means to circumvent size limitations on regular searches; the keyword *disabled* disables the control, i.e. no paged results can be returned. Note that the total number of entries returned when the pagedResults control is requested cannot exceed the **hard** size limit of regular searches unless extended by the **prtotal** switch.

The **limits** statement is typically used to let an unlimited number of entries be returned by searches performed with the identity used by the consumer for synchronization purposes by means of the RFC 4533 LDAP Content Synchronization protocol (see **syncrepl** for details).

maxderefdepth <depth>
Specifies the maximum number of aliases to dereference when trying to resolve an entry, used to avoid infinite alias loops. The default is 15.

mirrormode on | off
This option puts a replica database into "mirror" mode. Update operations will be accepted from any user, not just the updatedn. The database must already be configured as a syncrepl consumer before this keyword may be set. This mode also requires a **serverID** (see above) to be configured. By default, mirrormode is off.

monitoring on | off
This option enables database-specific monitoring in the entry related to the current database in the "cn=Databases,cn=Monitor" subtree of the monitor database, if the monitor database is enabled. Currently, only the BDB and the HDB databases provide database-specific monitoring. The default depends on the backend type.

overlay <overlay-name>
Add the specified overlay to this database. An overlay is a piece of code that intercepts database operations in order to extend or change them. Overlays are pushed onto a stack over the database, and so they will execute in the reverse of the order in which they were configured and the database itself will receive control last of all. See the **slapd.overlays**(5) manual page for an overview of the available overlays. Note that all of the database's regular settings should be configured before any overlay settings.

readonly on | off
This option puts the database into "read-only" mode. Any attempts to modify the database will return an "unwilling to perform" error. By default, readonly is off.

replica uri=ldap[s]://<hostname>[:port]|host=<hostname>[:port] [starttls=yes|critical] [suffix=<suffix> [...]] bindmethod=simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [attrs[!]=<attr list>]
Specify a replication site for this database. Refer to the "OpenLDAP Administrator's Guide" for detailed information on setting up a replicated **slapd** directory service. Zero or more **suffix** instances can be used to select the subtrees that will be replicated (defaults to all the database). **host** is deprecated in favor of the **uri** option. **uri** allows the replica LDAP server to be specified as an LDAP URI. A **bindmethod** of **simple** requires the options **binddn** and **credentials** and should only be used when adequate security services (e.g TLS or IPSEC) are in place. A **bindmethod** of **sasl** requires the option **saslmech.** Specific security properties (as with the **sasl-secprops** keyword above) for a SASL bind can be set with the **secprops** option. A non-default SASL realm can be set with the **realm** option. If the **mechanism** will use Kerberos, a kerberos instance should be given in **authcId.** An **attr list** can be given after the **attrs** keyword to allow the selective replication of the listed attributes only; if the optional **!** mark is used, the list is considered exclusive, i.e. the listed attributes are not replicated. If an objectClass is listed, all the related attributes are (are not) replicated.

restrict <oplist>

Specify a whitespace separated list of operations that are restricted. If defined inside a database specification, restrictions apply only to that database, otherwise they are global. Operations can be any of **add**, **bind**, **compare**, **delete**, **extended[=<OID>]**, **modify**, **rename**, **search**, or the special pseudo-operations **read** and **write**, which respectively summarize read and write operations. The use of *restrict write* is equivalent to *readonly on* (see above). The **extended** keyword allows to indicate the OID of the specific operation to be restricted.

rootdn <dn>

Specify the distinguished name that is not subject to access control or administrative limit restrictions for operations on this database. This DN may or may not be associated with an entry. An empty root DN (the default) specifies no root access is to be granted. It is recommended that the rootdn only be specified when needed (such as when initially populating a database). If the rootdn is within a namingContext (suffix) of the database, a simple bind password may also be provided using the **rootpw** directive. Many optional features, including syncrepl, require the rootdn to be defined for the database.

rootpw <password>

Specify a password (or hash of the password) for the rootdn. The password can only be set if the rootdn is within the namingContext (suffix) of the database. This option accepts all RFC 2307 userPassword formats known to the server (see **password-hash** description) as well as cleartext. **slappasswd**(8) may be used to generate a hash of a password. Cleartext and **{CRYPT}** passwords are not recommended. If empty (the default), authentication of the root DN is by other means (e.g. SASL). Use of SASL is encouraged.

suffix <dn suffix>

Specify the DN suffix of queries that will be passed to this backend database. Multiple suffix lines can be given and at least one is required for each database definition. If the suffix of one database is "inside" that of another, the database with the inner suffix must come first in the configuration file.

subordinate [advertise]

Specify that the current backend database is a subordinate of another backend database. A subordinate database may have only one suffix. This option may be used to glue multiple databases into a single namingContext. If the suffix of the current database is within the namingContext of a superior database, searches against the superior database will be propagated to the subordinate as well. All of the databases associated with a single namingContext should have identical rootdns. Behavior of other LDAP operations is unaffected by this setting. In particular, it is not possible to use moddn to move an entry from one subordinate to another subordinate within the namingContext.

If the optional **advertise** flag is supplied, the naming context of this database is advertised in the root DSE. The default is to hide this database context, so that only the superior context is visible.

If the slap tools **slapcat**(8), **slapadd**(8), or **slapindex**(8) are used on the superior database, any glued subordinates that support these tools are opened as well.

Databases that are glued together should usually be configured with the same indices (assuming they support indexing), even for attributes that only exist in some of these databases. In general, all of the glued databases should be configured as similarly as possible, since the intent is to provide the appearance of a single directory.

Note that the *subordinate* functionality is implemented internally by the *glue* overlay and as such its behavior will interact with other overlays in use. By default, the glue overlay is automatically configured as the last overlay on the superior backend. Its position on the backend can be explicitly configured by setting an **overlay glue** directive at the desired position. This explicit configuration is necessary e.g. when using the *syncprov* overlay, which needs to follow *glue* in

order to work over all of the glued databases. E.g.

```
database bdb
suffix dc=example,dc=com
...
overlay glue
overlay syncprov
```

syncrepl rid=<replica ID> provider=ldap[s]://<hostname>[:port] searchbase=<base DN> [type=refreshOnly|refreshAndPersist] [interval=dd:hh:mm:ss] [retry=[<retry interval> <# of retries>]+] [filter=<filter str>] [scope=sub|one|base|subord] [attrs=<attr list>] [attrsonly] [sizelimit=<limit>] [timelimit=<limit>] [schemachecking=on|off] [bindmethod=simple|sasl] [binddn=<dn>] [saslmech=<mech>] [authcid=<identity>] [authzid=<identity>] [credentials=<passwd>] [realm=<realm>] [secprops=<properties>] [starttls=yes|critical] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all] [logbase=<base DN>] [logfilter=<filter str>] [syncdata=default|accesslog|changelog]

Specify the current database as a replica which is kept up-to-date with the master content by establishing the current **slapd**(8) as a replication consumer site running a **syncrepl** replication engine. The replica content is kept synchronized to the master content using the LDAP Content Synchronization protocol. Refer to the "OpenLDAP Administrator's Guide" for detailed information on setting up a replicated **slapd** directory service using the **syncrepl** replication engine.

rid identifies the current **syncrepl** directive within the replication consumer site. It is a non-negative integer not greater than 4095 (limited to three hexadecimal digits).

provider specifies the replication provider site containing the master content as an LDAP URI. If <port> is not given, the standard LDAP port number (389 or 636) is used.

The content of the **syncrepl** replica is defined using a search specification as its result set. The consumer **slapd** will send search requests to the provider **slapd** according to the search specification. The search specification includes **searchbase**, **scope**, **filter**, **attrs**, **attrsonly**, **sizelimit**, and **timelimit** parameters as in the normal search specification. The **scope** defaults to **sub**, the **filter** defaults to **(objectclass=*)**, while there is no default **searchbase**. The **attrs** list defaults to **"*,+"** to return all user and operational attributes, and **attrsonly** is unset by default. The **sizelimit** and **timelimit** only accept "unlimited" and positive integers, and both default to "unlimited". The **sizelimit** and **timelimit** parameters define a consumer requested limitation on the number of entries that can be returned by the LDAP Content Synchronization operation; as such, it is intended to implement partial replication based on the size of the replicated database and on the time required by the synchronization. Note, however, that any provider-side limits for the replication identity will be enforced by the provider regardless of the limits requested by the LDAP Content Synchronization operation, much like for any other search operation.

The LDAP Content Synchronization protocol has two operation types. In the **refreshOnly** operation, the next synchronization search operation is periodically rescheduled at an interval time (specified by **interval** parameter; 1 day by default) after each synchronization operation finishes. In the **refreshAndPersist** operation, a synchronization search remains persistent in the provider slapd. Further updates to the master replica will generate **searchResultEntry** to the consumer slapd as the search responses to the persistent synchronization search.

If an error occurs during replication, the consumer will attempt to reconnect according to the **retry** parameter which is a list of the <retry interval> and <# of retries> pairs. For example, retry="60 10 300 3" lets the consumer retry every 60 seconds for the first 10 times and then retry every 300 seconds for the next 3 times before stop retrying. The '+' in <# of retries> means indefinite

number of retries until success.

The schema checking can be enforced at the LDAP Sync consumer site by turning on the **schemachecking** parameter. The default is **off**. Schema checking **on** means that replicated entries must have a structural objectClass, must obey to objectClass requirements in terms of required/allowed attributes, and that naming attributes and distinguished values must be present. As a consequence, schema checking should be **off** when partial replication is used.

A **bindmethod** of **simple** requires the options **binddn** and **credentials** and should only be used when adequate security services (e.g. TLS or IPSEC) are in place. **REMEMBER: simple bind credentials must be in cleartext!** A **bindmethod** of **sasl** requires the option **saslmech.** Depending on the mechanism, an authentication identity and/or credentials can be specified using **authcid** and **credentials.** The **authzid** parameter may be used to specify an authorization identity. Specific security properties (as with the **sasl-secprops** keyword above) for a SASL bind can be set with the **secprops** option. A non default SASL realm can be set with the **realm** option. The identity used for synchronization by the consumer should be allowed to receive an unlimited number of entries in response to a search request. The provider, other than allow authentication of the syncrepl identity, should grant that identity appropriate access privileges to the data that is being replicated (**access** directive), and appropriate time and size limits. This can be accomplished by either allowing unlimited **sizelimit** and **timelimit**, or by setting an appropriate **limits** statement in the consumer's configuration (see **sizelimit** and **limits** for details).

The **starttls** parameter specifies use of the StartTLS extended operation to establish a TLS session before Binding to the provider. If the **critical** argument is supplied, the session will be aborted if the StartTLS request fails. Otherwise the syncrepl session continues without TLS. The tls_reqcert setting defaults to "demand" and the other TLS settings default to the same as the main slapd TLS settings.

Rather than replicating whole entries, the consumer can query logs of data modifications. This mode of operation is referred to as *delta syncrepl*. In addition to the above parameters, the **logbase** and **logfilter** parameters must be set appropriately for the log that will be used. The **syncdata** parameter must be set to either "accesslog" if the log conforms to the **slapo-accesslog**(5) log format, or "changelog" if the log conforms to the obsolete *changelog* format. If the **syncdata** parameter is omitted or set to "default" then the log parameters are ignored.

updatedn <dn>

This option is only applicable in a slave database. It specifies the DN permitted to update (subject to access controls) the replica. It is only needed in certain push-mode replication scenarios. Generally, this DN *should not* be the same as the **rootdn** used at the master.

updateref <url>

Specify the referral to pass back when **slapd**(8) is asked to modify a replicated local database. If specified multiple times, each url is provided.

DATABASE-SPECIFIC OPTIONS

Each database may allow specific configuration options; they are documented separately in the backends' manual pages. See the **slapd.backends**(5) manual page for an overview of available backends.

EXAMPLES

Here is a short example of a configuration file:

```
include   SYSCONFDIR/schema/core.schema
pidfile   LOCALSTATEDIR/run/slapd.pid

# Subtypes of "name" (e.g. "cn" and "ou") with the
# option ";x-hidden" can be searched for/compared,
```

```
# but are not shown.  See slapd.access(5).
attributeoptions x-hidden lang-
access to attrs=name;x-hidden by * =cs

# Protect passwords.  See slapd.access(5).
access    to attrs=userPassword  by * auth
# Read access to other attributes and entries.
access    to *  by * read

database  bdb
suffix    "dc=our-domain,dc=com"
# The database directory MUST exist prior to
# running slapd AND should only be accessible
# by the slapd/tools. Mode 0700 recommended.
directory LOCALSTATEDIR/openldap-data
# Indices to maintain
index     objectClass  eq
index     cn,sn,mail   pres,eq,approx,sub

# We serve small clients that do not handle referrals,
# so handle remote lookups on their behalf.
database  ldap
suffix    ""
uri       ldap://ldap.some-server.com/
lastmod   off
```

"OpenLDAP Administrator's Guide" contains a longer annotated example of a configuration file. The original ETCDIR/slapd.conf is another example.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

ldap(3), **slapd−config**(5), **slapd.access**(5), **slapd.backends**(5), **slapd.overlays**(5), **slapd.plugin**(5), **slapd.replog**(5), **slapd**(8), **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), **slappasswd**(8), **slaptest**(8).

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

NAME

slapd.conf − configuration file for slapd, the stand-alone LDAP daemon

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The file **/etc/openldap/slapd.conf** contains configuration information for the **slapd**(8) daemon. This configuration file is also used by the SLAPD tools **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), and **slaptest**(8).

The **slapd.conf** file consists of a series of global configuration options that apply to **slapd** as a whole (including all backends), followed by zero or more database backend definitions that contain information specific to a backend instance. The configuration options are case-insensitive; their value, on a case by case basis, may be case-sensitive.

The general format of **slapd.conf** is as follows:

```
# comment - these options apply to every database
<global configuration options>
# first database definition & configuration options
database <backend 1 type>
<configuration options specific to backend 1>
# subsequent database definitions & configuration options
...
```

As many backend-specific sections as desired may be included. Global options can be overridden in a backend (for options that appear more than once, the last appearance in the **slapd.conf** file is used).

If a line begins with white space, it is considered a continuation of the previous line. No physical line should be over 2000 bytes long.

Blank lines and comment lines beginning with a '#' character are ignored. Note: continuation lines are unwrapped before comment processing is applied.

Arguments on configuration lines are separated by white space. If an argument contains white space, the argument should be enclosed in double quotes. If an argument contains a double quote ('"') or a backslash character ('\'), the character should be preceded by a backslash character.

The specific configuration options available are discussed below in the Global Configuration Options, General Backend Options, and General Database Options. Backend-specific options are discussed in the **slapd-<backend>(5)** manual pages. Refer to the "OpenLDAP Administrator's Guide" for more details on the slapd configuration file.

GLOBAL CONFIGURATION OPTIONS

Options described in this section apply to all backends, unless specifically overridden in a backend definition. Arguments that should be replaced by actual text are shown in brackets <>.

access to <what> [by <who> <access> <control>]+

Grant access (specified by <access>) to a set of entries and/or attributes (specified by <what>) by one or more requestors (specified by <who>). If no access controls are present, the default policy allows anyone and everyone to read anything but restricts updates to rootdn. (e.g., "access to * by * read"). The rootdn can always read and write EVERYTHING! See **slapd.access**(5) and the "OpenLDAP's Administrator's Guide" for details.

allow <features>

Specify a set of features (separated by white space) to allow (default none). **bind_v2** allows acceptance of LDAPv2 bind requests. Note that **slapd**(8) does not truly implement LDAPv2 (RFC 1777), now Historic (RFC 3494). **bind_anon_cred** allows anonymous bind when credentials are not empty (e.g. when DN is empty). **bind_anon_dn** allows unauthenticated (anonymous) bind when DN is not empty. **update_anon** allows unauthenticated (anonymous) update operations to be processed (subject to access controls and other administrative limits). **proxy_authz_anon**

allows unauthenticated (anonymous) proxy authorization control to be processed (subject to access controls, authorization and other administrative limits).

argsfile <filename>
The (absolute) name of a file that will hold the **slapd** server's command line options if started without the debugging command line option.

attributeoptions [option-name]...
Define tagging attribute options or option tag/range prefixes. Options must not end with '-', prefixes must end with '-'. The 'lang-' prefix is predefined. If you use the **attributeoptions** directive, 'lang-' will no longer be defined and you must specify it explicitly if you want it defined.

An attribute description with a tagging option is a subtype of that attribute description without the option. Except for that, options defined this way have no special semantics. Prefixes defined this way work like the 'lang-' options: They define a prefix for tagging options starting with the prefix. That is, if you define the prefix 'x-foo-', you can use the option 'x-foo-bar'. Furthermore, in a search or compare, a prefix or range name (with a trailing '-') matches all options starting with that name, as well as the option with the range name sans the trailing '-'. That is, 'x-foo-bar-' matches 'x-foo-bar' and 'x-foo-bar-baz'.

RFC 4520 reserves options beginning with 'x-' for private experiments. Other options should be registered with IANA, see RFC 4520 section 3.5. OpenLDAP also has the 'binary' option built in, but this is a transfer option, not a tagging option.

attributetype (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oid>] [EQUALITY <oid>] [ORDERING <oid>] [SUBSTR <oid>] [SYNTAX <oidlen>] [SINGLE−VALUE] [COLLECTIVE] [NO−USER−MODIFICATION] [USAGE <attributeUsage>])
Specify an attribute type using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **objectidentifier** description.)

authz-policy <policy>
Used to specify which rules to use for Proxy Authorization. Proxy authorization allows a client to authenticate to the server using one user's credentials, but specify a different identity to use for authorization and access control purposes. It essentially allows user A to login as user B, using user A's password. The **none** flag disables proxy authorization. This is the default setting. The **from** flag will use rules in the *authzFrom* attribute of the authorization DN. The **to** flag will use rules in the *authzTo* attribute of the authentication DN. The **any** flag, an alias for the deprecated value of **both**, will allow any of the above, whatever succeeds first (checked in **to**, **from** sequence. The **all** flag requires both authorizations to succeed.

The rules are mechanisms to specify which identities are allowed to perform proxy authorization. The *authzFrom* attribute in an entry specifies which other users are allowed to proxy login to this entry. The *authzTo* attribute in an entry specifies which other users this user can authorize as. Use of *authzTo* rules can be easily abused if users are allowed to write arbitrary values to this attribute. In general the *authzTo* attribute must be protected with ACLs such that only privileged users can modify it. The value of *authzFrom* and *authzTo* describes an **identity** or a set of identities; it can take five forms:

ldap:///<base>??[<scope>]?<filter>
dn[.<dnstyle>]:<pattern>
u[<mech>[<realm>]]:<pattern>
group[/objectClass[/attributeType]]:<pattern>
<pattern>

<dnstyle>:={exact|onelevel|children|subtree|regex}

The first form is a valid LDAP **URI** where the *<host>:<port>*, the *<attrs>* and the *<extensions>* portions must be absent, so that the search occurs locally on either *authzFrom* or *authzTo*. The second form is a **DN**, with the optional style modifiers *exact*, *onelevel*, *children*, and *subtree* for exact, onelevel, children and subtree matches, which cause *<pattern>* to be normalized according to the DN normalization rules, or the special *regex* style, which causes the *<pattern>* to be treated as a POSIX (''extended'') regular expression, as discussed in **regex**(7) and/or **re_format**(7). A pattern of * means any non-anonymous DN. The third form is a SASL **id**, with the optional fields *<mech>* and *<realm>* that allow to specify a SASL **mechanism**, and eventually a SASL **realm**, for those mechanisms that support one. The need to allow the specification of a mechanism is still debated, and users are strongly discouraged to rely on this possibility. The fourth form is a group specification, consisting of the keyword **group**, optionally followed by the specification of the group **objectClass** and member **attributeType**. The group with DN **<pattern>** is searched with base scope, and in case of match, the values of the member **attributeType** are searched for the asserted DN. For backwards compatibility, if no identity type is provided, i.e. only **<pattern>** is present, an *exact DN* is assumed; as a consequence, **<pattern>** is subjected to DN normalization. Since the interpretation of *authzFrom* and *authzTo* can impact security, users are strongly encouraged to explicitly set the type of identity specification that is being used. A subset of these rules can be used as third arg in the **authz-regexp** statement (see below); significantly, the *URI* and the *dn.exact:<dn>* forms.

authz-regexp <match> <replace>

Used by the authentication framework to convert simple user names, such as provided by SASL subsystem, to an LDAP DN used for authorization purposes. Note that the resultant DN need not refer to an existing entry to be considered valid. When an authorization request is received from the SASL subsystem, the SASL **USERNAME**, **REALM**, and **MECHANISM** are taken, when available, and combined into a name of the form

UID=<username>[[,CN=<realm>],CN=<mechanism>],CN=auth

This name is then compared against the **match** POSIX (''extended'') regular expression, and if the match is successful, the name is replaced with the **replace** string. If there are wildcard strings in the **match** regular expression that are enclosed in parenthesis, e.g.

UID=([^,]*),CN=.*

then the portion of the name that matched the wildcard will be stored in the numbered placeholder variable $1. If there are other wildcard strings in parenthesis, the matching strings will be in $2, $3, etc. up to $9. The placeholders can then be used in the **replace** string, e.g.

UID=$1,OU=Accounts,DC=example,DC=com

The replaced name can be either a DN, i.e. a string prefixed by "dn:", or an LDAP URI. If the latter, the server will use the URI to search its own database(s) and, if the search returns exactly one entry, the name is replaced by the DN of that entry. The LDAP URI must have no hostport, attrs, or extensions components, but the filter is mandatory, e.g.

ldap:///OU=Accounts,DC=example,DC=com??one?(UID=$1)

The protocol portion of the URI must be strictly **ldap**. Note that this search is subject to access controls. Specifically, the authentication identity must have "auth" access in the subject.

Multiple **authz-regexp** options can be given in the configuration file to allow for multiple matching and replacement patterns. The matching patterns are checked in the order they appear in the file, stopping at the first successful match.

concurrency <integer>
Specify a desired level of concurrency. Provided to the underlying thread system as a hint. The default is not to provide any hint.

conn_max_pending <integer>
Specify the maximum number of pending requests for an anonymous session. If requests are submitted faster than the server can process them, they will be queued up to this limit. If the limit is exceeded, the session is closed. The default is 100.

conn_max_pending_auth <integer>
Specify the maximum number of pending requests for an authenticated session. The default is 1000.

defaultsearchbase <dn>
Specify a default search base to use when client submits a non-base search request with an empty base DN. Base scoped search requests with an empty base DN are not affected.

disallow <features>
Specify a set of features (separated by white space) to disallow (default none). **bind_anon** disables acceptance of anonymous bind requests. Note that this setting does not prohibit anonymous directory access (See "require authc"). **bind_simple** disables simple (bind) authentication. **tls_2_anon** disables forcing session to anonymous status (see also **tls_authc**) upon StartTLS operation receipt. **tls_authc** disallows the StartTLS operation if authenticated (see also **tls_2_anon**).

ditcontentrule (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [AUX <oids>] [MUST <oids>] [MAY <oids>] [NOT <oids>])
Specify an DIT Content Rule using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the attribute OID and attribute syntax OID. (See the **objectidentifier** description.)

gentlehup { on | off }
A SIGHUP signal will only cause a 'gentle' shutdown-attempt: **Slapd** will stop listening for new connections, but will not close the connections to the current clients. Future write operations return unwilling-to-perform, though. Slapd terminates when all clients have closed their connections (if they ever do), or − as before − if it receives a SIGTERM signal. This can be useful if you wish to terminate the server and start a new **slapd** server **with another database,** without disrupting the currently active clients. The default is off. You may wish to use **idletimeout** along with this option.

idletimeout <integer>
Specify the number of seconds to wait before forcibly closing an idle client connection. A idletimeout of 0 disables this feature. The default is 0.

include <filename>
Read additional configuration information from the given file before continuing with the next line of the current file.

index_intlen <integer>
Specify the key length for ordered integer indices. The most significant bytes of the binary integer will be used for index keys. The default value is 4, which provides exact indexing for 31 bit values. A floating point representation is used to index too large values.

index_substr_if_minlen <integer>
Specify the minimum length for subinitial and subfinal indices. An attribute value must have at least this many characters in order to be processed by the indexing functions. The default is 2.

index_substr_if_maxlen <integer>
Specify the maximum length for subinitial and subfinal indices. Only this many characters of an attribute value will be processed by the indexing functions; any excess characters are ignored. The default is 4.

index_substr_any_len <integer>
Specify the length used for subany indices. An attribute value must have at least this many characters in order to be processed. Attribute values longer than this length will be processed in segments of this length. The default is 4. The subany index will also be used in subinitial and subfinal index lookups when the filter string is longer than the *index_substr_if_maxlen* value.

index_substr_any_step <integer>
Specify the steps used in subany index lookups. This value sets the offset for the segments of a filter string that are processed for a subany index lookup. The default is 2. For example, with the default values, a search using this filter "cn=*abcdefgh*" would generate index lookups for "abcd", "cdef", and "efgh".

Note: Indexing support depends on the particular backend in use. Also, changing these settings will generally require deleting any indices that depend on these parameters and recreating them with **slapindex**(8).

localSSF <SSF>
Specifies the Security Strength Factor (SSF) to be given local LDAP sessions, such as those to the ldapi:// listener. For a description of SSF values, see **sasl-secprops**'s **minssf** option description. The default is 71.

logfile <filename>
Specify a file for recording debug log messages. By default these messages only go to stderr and are not recorded anywhere else. Specifying a logfile copies messages to both stderr and the logfile.

loglevel <integer> [...]
Specify the level at which debugging statements and operation statistics should be syslogged (currently logged to the **syslogd**(8) LOG_LOCAL4 facility). They must be considered subsystems rather than increasingly verbose log levels. Some messages with higher priority are logged regardless of the configured loglevel as soon as any logging is configured. Log levels are additive, and available levels are:

1	**(0x1 trace)** trace function calls
2	**(0x2 packets)** debug packet handling
4	**(0x4 args)** heavy trace debugging (function args)
8	**(0x8 conns)** connection management
16	**(0x10 BER)** print out packets sent and received
32	**(0x20 filter)** search filter processing
64	**(0x40 config)** configuration file processing
128	**(0x80 ACL)** access control list processing
256	**(0x100 stats)** connections, LDAP operations, results (recommended)
512	**(0x200 stats2)** stats log entries sent
1024	**(0x400 shell)** print communication with shell backends
2048	**(0x800 parse)** entry parsing
16384	**(0x4000 sync)** LDAPSync replication
32768	**(0x8000 none)** only messages that get logged whatever log level is set

The desired log level can be input as a single integer that combines the (ORed) desired levels, both in decimal or in hexadecimal notation, as a list of integers (that are ORed internally), or as a list of

the names that are shown between brackets, such that

loglevel 129
loglevel 0x81
loglevel 128 1
loglevel 0x80 0x1
loglevel acl trace

are equivalent. The keyword **any** can be used as a shortcut to enable logging at all levels (equivalent to -1). The keyword **none**, or the equivalent integer representation, causes those messages that are logged regardless of the configured loglevel to be logged. In fact, if loglevel is set to 0, no logging occurs, so at least the **none** level is required to have high priority messages logged.

The loglevel defaults to **stats**. This level should usually also be included when using other loglevels, to help analyze the logs.

moduleload <filename>
Specify the name of a dynamically loadable module to load. The filename may be an absolute path name or a simple filename. Non-absolute names are searched for in the directories specified by the **modulepath** option. This option and the **modulepath** option are only usable if slapd was compiled with --enable-modules.

modulepath <pathspec>
Specify a list of directories to search for loadable modules. Typically the path is colon-separated but this depends on the operating system.

objectclass (<oid> [NAME <name>] [DESC <description>] [OBSOLETE] [SUP <oids>] [{ ABSTRACT | STRUCTURAL | AUXILIARY }] [MUST <oids>] [MAY <oids>])
Specify an objectclass using the LDAPv3 syntax defined in RFC 4512. The slapd parser extends the RFC 4512 definition by allowing string forms as well as numeric OIDs to be used for the object class OID. (See the **objectidentifier** description.) Object classes are "STRUCTURAL" by default.

objectidentifier <name> { <oid> | <name>[:<suffix>] }
Define a string name that equates to the given OID. The string can be used in place of the numeric OID in objectclass and attribute definitions. The name can also be used with a suffix of the form ":xx" in which case the value "oid.xx" will be used.

password-hash <hash> [<hash>...]
This option configures one or more hashes to be used in generation of user passwords stored in the userPassword attribute during processing of LDAP Password Modify Extended Operations (RFC 3062). The <hash> must be one of **{SSHA}**, **{SHA}**, **{SMD5}**, **{MD5}**, **{CRYPT}**, and **{CLEARTEXT}**. The default is **{SSHA}**.

{SHA} and **{SSHA}** use the SHA-1 algorithm (FIPS 160-1), the latter with a seed.

{MD5} and **{SMD5}** use the MD5 algorithm (RFC 1321), the latter with a seed.

{CRYPT} uses the **crypt**(3).

{CLEARTEXT} indicates that the new password should be added to userPassword as clear text.

Note that this option does not alter the normal user applications handling of userPassword during LDAP Add, Modify, or other LDAP operations.

password−crypt−salt−format <format>
Specify the format of the salt passed to **crypt**(3) when generating {CRYPT} passwords (see **password−hash**) during processing of LDAP Password Modify Extended Operations (RFC 3062).

This string needs to be in **sprintf**(3) format and may include one (and only one) %s conversion. This conversion will be substituted with a string of random characters from [A−Za−z0−9./]. For example, "%.2s" provides a two character salt and "1%.8s" tells some versions of crypt(3) to use an MD5 algorithm and provides 8 random characters of salt. The default is "%s", which provides 31 characters of salt.

pidfile <filename>

The (absolute) name of a file that will hold the **slapd** server's process ID (see **getpid**(2)) if started without the debugging command line option.

referral <url>

Specify the referral to pass back when **slapd**(8) cannot find a local database to handle a request. If specified multiple times, each url is provided.

require <conditions>

Specify a set of conditions (separated by white space) to require (default none). The directive may be specified globally and/or per-database; databases inherit global conditions, so per-database specifications are additive. **bind** requires bind operation prior to directory operations. **LDAPv3** requires session to be using LDAP version 3. **authc** requires authentication prior to directory operations. **SASL** requires SASL authentication prior to directory operations. **strong** requires strong authentication prior to directory operations. The strong keyword allows protected "simple" authentication as well as SASL authentication. **none** may be used to require no conditions (useful to clear out globally set conditions within a particular database); it must occur first in the list of conditions.

reverse-lookup on | off

Enable/disable client name unverified reverse lookup (default is **off** if compiled with --enable-rlookups).

rootDSE <file>

Specify the name of an LDIF(5) file containing user defined attributes for the root DSE. These attributes are returned in addition to the attributes normally produced by slapd.

The root DSE is an entry with information about the server and its capabilities, in operational attributes. It has the empty DN, and can be read with e.g.:

```
ldapsearch -x -b "" -s base "+"
```

See RFC 4512 section 5.1 for details.

sasl-host <fqdn>

Used to specify the fully qualified domain name used for SASL processing.

sasl-realm <realm>

Specify SASL realm. Default is empty.

sasl-secprops <properties>

Used to specify Cyrus SASL security properties. The **none** flag (without any other properties) causes the flag properties default, "noanonymous,noplain", to be cleared. The **noplain** flag disables mechanisms susceptible to simple passive attacks. The **noactive** flag disables mechanisms susceptible to active attacks. The **nodict** flag disables mechanisms susceptible to passive dictionary attacks. The **noanonymous** flag disables mechanisms which support anonymous login. The **forwardsec** flag require forward secrecy between sessions. The **passcred** require mechanisms which pass client credentials (and allow mechanisms which can pass credentials to do so). The **minssf=<factor>** property specifies the minimum acceptable *security strength factor* as an integer approximate to effective key length used for encryption. 0 (zero) implies no protection, 1 implies integrity protection only, 56 allows DES or other weak ciphers, 112 allows triple DES and other strong ciphers, 128 allows RC4, Blowfish and other modern strong ciphers. The default is 0. The **maxssf=<factor>** property specifies the maximum acceptable *security strength factor* as an integer (see minssf description). The default is INT_MAX. The **maxbufsize=<size>** property specifies the maximum security layer receive

buffer size allowed. 0 disables security layers. The default is 65536.

schemadn <dn>
Specify the distinguished name for the subschema subentry that controls the entries on this server. The default is "cn=Subschema".

security <factors>
Specify a set of security strength factors (separated by white space) to require (see **sasl-secprops**'s **minssf** option for a description of security strength factors). The directive may be specified globally and/or per-database. **ssf=<n>** specifies the overall security strength factor. **transport=<n>** specifies the transport security strength factor. **tls=<n>** specifies the TLS security strength factor. **sasl=<n>** specifies the SASL security strength factor. **update_ssf=<n>** specifies the overall security strength factor to require for directory updates. **update_transport=<n>** specifies the transport security strength factor to require for directory updates. **update_tls=<n>** specifies the TLS security strength factor to require for directory updates. **update_sasl=<n>** specifies the SASL security strength factor to require for directory updates. **simple_bind=<n>** specifies the security strength factor required for *simple* username/password authentication. Note that the **transport** factor is measure of security provided by the underlying transport, e.g. ldapi:// (and eventually IPSEC). It is not normally used.

serverID <integer> [<URL>]
Specify an integer ID from 0 to 4095 for this server (limited to 3 hexadecimal digits). These IDs are required when using multimaster replication and each master must have a unique ID. If the URL is provided, this directive may be specified multiple times, providing a complete list of participating servers and their IDs. The fully qualified hostname of each server should be used in the supplied URLs. The IDs are used in the "replica id" field of all CSNs generated by the specified server. The default value is zero. Example:

serverID 1

sizelimit {<integer>|unlimited}

sizelimit size[.{soft|hard|unchecked}]=<integer> [...]
Specify the maximum number of entries to return from a search operation. The default size limit is 500. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the size limits. Extra args can be added on the same line. See **limits** for an explanation of the different flags.

sockbuf_max_incoming <integer>
Specify the maximum incoming LDAP PDU size for anonymous sessions. The default is 262143.

sockbuf_max_incoming_auth <integer>
Specify the maximum incoming LDAP PDU size for authenticated sessions. The default is 4194303.

sortvals <attr> [...]
Specify a list of multi-valued attributes whose values will always be maintained in sorted order. Using this option will allow Modify, Compare, and filter evaluations on these attributes to be performed more efficiently. The resulting sort order depends on the attributes' syntax and matching rules and may not correspond to lexical order or any other recognizable order.

threads <integer>
Specify the maximum size of the primary thread pool. The default is 16; the minimum value is 2.

timelimit {<integer>|unlimited}

timelimit time[.{soft|hard}]=<integer> [...]
Specify the maximum number of seconds (in real time) **slapd** will spend answering a search request. The default time limit is 3600. Use **unlimited** to specify no limits. The second format allows a fine grain setting of the time limits. Extra args can be added on the same line. See **limits** for an explanation of the different flags.

tool-threads <integer>
Specify the maximum number of threads to use in tool mode. This should not be greater than the number of CPUs in the system. The default is 1.

TLS OPTIONS

If **slapd** is built with support for Transport Layer Security, there are more options you can specify.

TLSCipherSuite <cipher-suite-spec>
Permits configuring what ciphers will be accepted and the preference order. <cipher-suite-spec> should be a cipher specification for OpenSSL. Example:

TLSCipherSuite HIGH:MEDIUM:+SSLv2

To check what ciphers a given spec selects, use:

openssl ciphers -v <cipher-suite-spec>

To obtain the list of ciphers in GNUtls use:

gnutls-cli -l

TLSCACertificateFile <filename>
Specifies the file that contains certificates for all of the Certificate Authorities that **slapd** will recognize.

TLSCACertificatePath <path>
Specifies the path of a directory that contains Certificate Authority certificates in separate individual files. Usually only one of this or the TLSCACertificateFile is used. This directive is not supported when using GNUtls.

TLSCertificateFile <filename>
Specifies the file that contains the **slapd** server certificate.

TLSCertificateKeyFile <filename>
Specifies the file that contains the **slapd** server private key that matches the certificate stored in the **TLSCertificateFile** file. Currently, the private key must not be protected with a password, so it is of critical importance that it is protected carefully.

TLSDHParamFile <filename>
This directive specifies the file that contains parameters for Diffie-Hellman ephemeral key exchange. This is required in order to use a DSA certificate on the server. If multiple sets of parameters are present in the file, all of them will be processed. Note that setting this option may also enable Anonymous Diffie-Hellman key exchanges in certain non-default cipher suites. You should append "!ADH" to your cipher suites if you have changed them from the default, otherwise no certificate exchanges or verification will be done. When using GNUtls these parameters are always generated randomly so this directive is ignored.

TLSRandFile <filename>
Specifies the file to obtain random bits from when /dev/[u]random is not available. Generally set to the name of the EGD/PRNGD socket. The environment variable RANDFILE can also be used to specify the filename. This directive is ignored with GNUtls.

TLSVerifyClient <level>
Specifies what checks to perform on client certificates in an incoming TLS session, if any. The **<level>** can be specified as one of the following keywords:

never This is the default. **slapd** will not ask the client for a certificate.

allow The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, it will be ignored and the session proceeds

normally.

try The client certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, the session is immediately terminated.

demand | hard | true
These keywords are all equivalent, for compatibility reasons. The client certificate is requested. If no certificate is provided, or a bad certificate is provided, the session is immediately terminated.

Note that a valid client certificate is required in order to use the SASL EXTERNAL authentication mechanism with a TLS session. As such, a non-default **TLSVerifyClient** setting must be chosen to enable SASL EXTERNAL authentication.

TLSCRLCheck <level>
Specifies if the Certificate Revocation List (CRL) of the CA should be used to verify if the client certificates have not been revoked. This requires **TLSCACertificatePath** parameter to be set. This directive is ignored with GNUtls. **<level>** can be specified as one of the following keywords:

none No CRL checks are performed

peer Check the CRL of the peer certificate

all Check the CRL for a whole certificate chain

TLSCRLFile <filename>
Specifies a file containing a Certificate Revocation List to be used for verifying that certificates have not been revoked. This directive is only valid when using GNUtls.

GENERAL BACKEND OPTIONS

Options in this section only apply to the configuration file section for the specified backend. They are supported by every type of backend.

backend <databasetype>
Mark the beginning of a backend definition. <databasetype> should be one of **bdb**, **config**, **dnssrv**, **hdb**, **ldap**, **ldif**, **meta**, **monitor**, **null**, **passwd**, **perl**, **relay**, **shell**, or **sql**, depending on which backend will serve the database.

GENERAL DATABASE OPTIONS

Options in this section only apply to the configuration file section for the database in which they are defined. They are supported by every type of backend. Note that the **database** and at least one **suffix** option are mandatory for each database.

database <databasetype>
Mark the beginning of a new database instance definition. <databasetype> should be one of **bdb**, **config**, **dnssrv**, **hdb**, **ldap**, **ldif**, **meta**, **monitor**, **null**, **passwd**, **perl**, **relay**, **shell**, or **sql**, depending on which backend will serve the database.

hidden on | off
Controls whether the database will be used to answer queries. A database that is hidden will never be selected to answer any queries, and any suffix configured on the database will be ignored in checks for conflicts with other databases. By default, hidden is off.

lastmod on | off
Controls whether **slapd** will automatically maintain the modifiersName, modifyTimestamp, creatorsName, and createTimestamp attributes for entries. It also controls the entryCSN and entryUUID attributes, which are needed by the syncrepl provider. By default, lastmod is on.

limits <who> <limit> [<limit> [...]]
Specify time and size limits based on who initiated an operation. The argument **who** can be any of

anonymous | users | [dn[.<style>]=]<pattern> | group[/oc[/at]]=<pattern>

with

<style> ::= exact | base | onelevel | subtree | children | regex | anonymous

The term **anonymous** matches all unauthenticated clients. The term **users** matches all authenticated clients; otherwise an **exact** dn pattern is assumed unless otherwise specified by qualifying the (optional) key string **dn** with **exact** or **base** (which are synonyms), to require an exact match; with **onelevel**, to require exactly one level of depth match; with **subtree**, to allow any level of depth match, including the exact match; with **children**, to allow any level of depth match, not including the exact match; **regex** explicitly requires the (default) match based on POSIX (''extended'') regular expression pattern. Finally, **anonymous** matches unbound operations; the **pattern** field is ignored. The same behavior is obtained by using the **anonymous** form of the **who** clause. The term **group**, with the optional objectClass **oc** and attributeType **at** fields, followed by **pattern**, sets the limits for any DN listed in the values of the **at** attribute (default **member**) of the **oc** group objectClass (default **groupOfNames**) whose DN exactly matches **pattern**.

The currently supported limits are **size** and **time**.

The syntax for time limits is **time[.{soft|hard}]=<integer>**, where *integer* is the number of seconds slapd will spend answering a search request. If no time limit is explicitly requested by the client, the **soft** limit is used; if the requested time limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for time limits smaller or equal to the **hard** limit are honored. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

The syntax for size limits is **size[.{soft|hard|unchecked}]=<integer>**, where *integer* is the maximum number of entries slapd will return answering a search request. If no size limit is explicitly requested by the client, the **soft** limit is used; if the requested size limit exceeds the **hard** limit, the value of the limit is used instead. If the **hard** limit is set to the keyword *soft*, the soft limit is used in either case; if it is set to the keyword *unlimited*, no hard limit is enforced. Explicit requests for size limits smaller or equal to the **hard** limit are honored. The **unchecked** specifier sets a limit on the number of candidates a search request is allowed to examine. The rationale behind it is that searches for non-properly indexed attributes may result in large sets of candidates, which must be examined by **slapd**(8) to determine whether they match the search filter or not. The **unchecked** limit provides a means to drop such operations before they are even started. If the selected candidates exceed the **unchecked** limit, the search will abort with *Unwilling to perform*. If it is set to the keyword *unlimited*, no limit is applied (the default). If it is set to *disable*, the search is not even performed; this can be used to disallow searches for a specific set of users. If no limit specifier is set, the value is assigned to the **soft** limit, and the **hard** limit is set to *soft*, to preserve the original behavior.

In case of no match, the global limits are used. The default values are the same of **sizelimit** and **timelimit**; no limit is set on **unchecked**.

If **pagedResults** control is requested, the **hard** size limit is used by default, because the request of a specific page size is considered an explicit request for a limitation on the number of entries to be returned. However, the size limit applies to the total count of entries returned within the search, and not to a single page. Additional size limits may be enforced; the syntax is **size.pr={<integer>|noEstimate|unlimited}**, where *integer* is the max page size if no explicit limit is set; the keyword *noEstimate* inhibits the server from returning an estimate of the total number of entries that might be returned (note: the current implementation does not return any estimate). The keyword *unlimited* indicates that no limit is applied to the pagedResults control page size. The

syntax **size.prtotal={<integer>|unlimited|disabled}** allows to set a limit on the total number of entries that a pagedResults control allows to return. By default it is set to the **hard** limit. When set, *integer* is the max number of entries that the whole search with pagedResults control can return. Use *unlimited* to allow unlimited number of entries to be returned, e.g. to allow the use of the pagedResults control as a means to circumvent size limitations on regular searches; the keyword *disabled* disables the control, i.e. no paged results can be returned. Note that the total number of entries returned when the pagedResults control is requested cannot exceed the **hard** size limit of regular searches unless extended by the **prtotal** switch.

The **limits** statement is typically used to let an unlimited number of entries be returned by searches performed with the identity used by the consumer for synchronization purposes by means of the RFC 4533 LDAP Content Synchronization protocol (see **syncrepl** for details).

maxderefdepth <depth>
Specifies the maximum number of aliases to dereference when trying to resolve an entry, used to avoid infinite alias loops. The default is 15.

mirrormode on | off
This option puts a replica database into "mirror" mode. Update operations will be accepted from any user, not just the updatedn. The database must already be configured as a syncrepl consumer before this keyword may be set. This mode also requires a **serverID** (see above) to be configured. By default, mirrormode is off.

monitoring on | off
This option enables database-specific monitoring in the entry related to the current database in the "cn=Databases,cn=Monitor" subtree of the monitor database, if the monitor database is enabled. Currently, only the BDB and the HDB databases provide database-specific monitoring. The default depends on the backend type.

overlay <overlay-name>
Add the specified overlay to this database. An overlay is a piece of code that intercepts database operations in order to extend or change them. Overlays are pushed onto a stack over the database, and so they will execute in the reverse of the order in which they were configured and the database itself will receive control last of all. See the **slapd.overlays**(5) manual page for an overview of the available overlays. Note that all of the database's regular settings should be configured before any overlay settings.

readonly on | off
This option puts the database into "read-only" mode. Any attempts to modify the database will return an "unwilling to perform" error. By default, readonly is off.

replica uri=ldap[s]://<hostname>[:port]|host=<hostname>[:port] [starttls=yes|critical] [suffix=<suffix> [...]] bindmethod=simple|sasl [binddn=<simple DN>] [credentials=<simple password>] [saslmech=<SASL mech>] [secprops=<properties>] [realm=<realm>] [authcId=<authentication ID>] [authzId=<authorization ID>] [attrs[!]=<attr list>]
Specify a replication site for this database. Refer to the "OpenLDAP Administrator's Guide" for detailed information on setting up a replicated **slapd** directory service. Zero or more **suffix** instances can be used to select the subtrees that will be replicated (defaults to all the database). **host** is deprecated in favor of the **uri** option. **uri** allows the replica LDAP server to be specified as an LDAP URI. A **bindmethod** of **simple** requires the options **binddn** and **credentials** and should only be used when adequate security services (e.g TLS or IPSEC) are in place. A **bindmethod** of **sasl** requires the option **saslmech.** Specific security properties (as with the **sasl-secprops** keyword above) for a SASL bind can be set with the **secprops** option. A non-default SASL realm can be set with the **realm** option. If the **mechanism** will use Kerberos, a kerberos instance should be given in **authcId.** An **attr list** can be given after the **attrs** keyword to allow the selective replication of the listed attributes only; if the optional **!** mark is used, the list is considered exclusive, i.e. the listed attributes are not replicated. If an objectClass is listed, all the related attributes are (are not) replicated.

restrict <oplist>

Specify a whitespace separated list of operations that are restricted. If defined inside a database specification, restrictions apply only to that database, otherwise they are global. Operations can be any of **add**, **bind**, **compare**, **delete**, **extended[=<OID>]**, **modify**, **rename**, **search**, or the special pseudo-operations **read** and **write**, which respectively summarize read and write operations. The use of *restrict write* is equivalent to *readonly on* (see above). The **extended** keyword allows to indicate the OID of the specific operation to be restricted.

rootdn <dn>

Specify the distinguished name that is not subject to access control or administrative limit restrictions for operations on this database. This DN may or may not be associated with an entry. An empty root DN (the default) specifies no root access is to be granted. It is recommended that the rootdn only be specified when needed (such as when initially populating a database). If the rootdn is within a namingContext (suffix) of the database, a simple bind password may also be provided using the **rootpw** directive. Many optional features, including syncrepl, require the rootdn to be defined for the database.

rootpw <password>

Specify a password (or hash of the password) for the rootdn. The password can only be set if the rootdn is within the namingContext (suffix) of the database. This option accepts all RFC 2307 userPassword formats known to the server (see **password-hash** description) as well as cleartext. **slappasswd**(8) may be used to generate a hash of a password. Cleartext and **{CRYPT}** passwords are not recommended. If empty (the default), authentication of the root DN is by other means (e.g. SASL). Use of SASL is encouraged.

suffix <dn suffix>

Specify the DN suffix of queries that will be passed to this backend database. Multiple suffix lines can be given and at least one is required for each database definition. If the suffix of one database is "inside" that of another, the database with the inner suffix must come first in the configuration file.

subordinate [advertise]

Specify that the current backend database is a subordinate of another backend database. A subordinate database may have only one suffix. This option may be used to glue multiple databases into a single namingContext. If the suffix of the current database is within the namingContext of a superior database, searches against the superior database will be propagated to the subordinate as well. All of the databases associated with a single namingContext should have identical rootdns. Behavior of other LDAP operations is unaffected by this setting. In particular, it is not possible to use moddn to move an entry from one subordinate to another subordinate within the namingContext.

If the optional **advertise** flag is supplied, the naming context of this database is advertised in the root DSE. The default is to hide this database context, so that only the superior context is visible.

If the slap tools **slapcat**(8), **slapadd**(8), or **slapindex**(8) are used on the superior database, any glued subordinates that support these tools are opened as well.

Databases that are glued together should usually be configured with the same indices (assuming they support indexing), even for attributes that only exist in some of these databases. In general, all of the glued databases should be configured as similarly as possible, since the intent is to provide the appearance of a single directory.

Note that the *subordinate* functionality is implemented internally by the *glue* overlay and as such its behavior will interact with other overlays in use. By default, the glue overlay is automatically configured as the last overlay on the superior backend. Its position on the backend can be explicitly configured by setting an **overlay glue** directive at the desired position. This explicit configuration is necessary e.g. when using the *syncprov* overlay, which needs to follow *glue* in

order to work over all of the glued databases. E.g.

```
database bdb
suffix dc=example,dc=com
...
overlay glue
overlay syncprov
```

syncrepl rid=<replica ID> provider=ldap[s]://<hostname>[:port] searchbase=<base DN> [type=refreshOnly|refreshAndPersist] [interval=dd:hh:mm:ss] [retry=[<retry interval> <# of retries>]+] [filter=<filter str>] [scope=sub|one|base|subord] [attrs=<attr list>] [attrsonly] [sizelimit=<limit>] [timelimit=<limit>] [schemachecking=on|off] [bindmethod=simple|sasl] [binddn=<dn>] [saslmech=<mech>] [authcid=<identity>] [authzid=<identity>] [credentials=<passwd>] [realm=<realm>] [secprops=<properties>] [starttls=yes|critical] [tls_cert=<file>] [tls_key=<file>] [tls_cacert=<file>] [tls_cacertdir=<path>] [tls_reqcert=never|allow|try|demand] [tls_ciphersuite=<ciphers>] [tls_crlcheck=none|peer|all] [logbase=<base DN>] [logfilter=<filter str>] [syncdata=default|accesslog|changelog]

Specify the current database as a replica which is kept up-to-date with the master content by establishing the current **slapd**(8) as a replication consumer site running a **syncrepl** replication engine. The replica content is kept synchronized to the master content using the LDAP Content Synchronization protocol. Refer to the "OpenLDAP Administrator's Guide" for detailed information on setting up a replicated **slapd** directory service using the **syncrepl** replication engine.

rid identifies the current **syncrepl** directive within the replication consumer site. It is a non-negative integer not greater than 4095 (limited to three hexadecimal digits).

provider specifies the replication provider site containing the master content as an LDAP URI. If <port> is not given, the standard LDAP port number (389 or 636) is used.

The content of the **syncrepl** replica is defined using a search specification as its result set. The consumer **slapd** will send search requests to the provider **slapd** according to the search specification. The search specification includes **searchbase**, **scope**, **filter**, **attrs**, **attrsonly**, **sizelimit**, and **timelimit** parameters as in the normal search specification. The **scope** defaults to **sub**, the **filter** defaults to **(objectclass=*)**, while there is no default **searchbase**. The **attrs** list defaults to **"*,+"** to return all user and operational attributes, and **attrsonly** is unset by default. The **sizelimit** and **timelimit** only accept "unlimited" and positive integers, and both default to "unlimited". The **sizelimit** and **timelimit** parameters define a consumer requested limitation on the number of entries that can be returned by the LDAP Content Synchronization operation; as such, it is intended to implement partial replication based on the size of the replicated database and on the time required by the synchronization. Note, however, that any provider-side limits for the replication identity will be enforced by the provider regardless of the limits requested by the LDAP Content Synchronization operation, much like for any other search operation.

The LDAP Content Synchronization protocol has two operation types. In the **refreshOnly** operation, the next synchronization search operation is periodically rescheduled at an interval time (specified by **interval** parameter; 1 day by default) after each synchronization operation finishes. In the **refreshAndPersist** operation, a synchronization search remains persistent in the provider slapd. Further updates to the master replica will generate **searchResultEntry** to the consumer slapd as the search responses to the persistent synchronization search.

If an error occurs during replication, the consumer will attempt to reconnect according to the **retry** parameter which is a list of the <retry interval> and <# of retries> pairs. For example, retry="60 10 300 3" lets the consumer retry every 60 seconds for the first 10 times and then retry every 300 seconds for the next 3 times before stop retrying. The '+' in <# of retries> means indefinite

number of retries until success.

The schema checking can be enforced at the LDAP Sync consumer site by turning on the **schemachecking** parameter. The default is **off**. Schema checking **on** means that replicated entries must have a structural objectClass, must obey to objectClass requirements in terms of required/allowed attributes, and that naming attributes and distinguished values must be present. As a consequence, schema checking should be **off** when partial replication is used.

A **bindmethod** of **simple** requires the options **binddn** and **credentials** and should only be used when adequate security services (e.g. TLS or IPSEC) are in place. **REMEMBER: simple bind credentials must be in cleartext!** A **bindmethod** of **sasl** requires the option **saslmech.** Depending on the mechanism, an authentication identity and/or credentials can be specified using **authcid** and **credentials.** The **authzid** parameter may be used to specify an authorization identity. Specific security properties (as with the **sasl-secprops** keyword above) for a SASL bind can be set with the **secprops** option. A non default SASL realm can be set with the **realm** option. The identity used for synchronization by the consumer should be allowed to receive an unlimited number of entries in response to a search request. The provider, other than allow authentication of the syncrepl identity, should grant that identity appropriate access privileges to the data that is being replicated (**access** directive), and appropriate time and size limits. This can be accomplished by either allowing unlimited **sizelimit** and **timelimit**, or by setting an appropriate **limits** statement in the consumer's configuration (see **sizelimit** and **limits** for details).

The **starttls** parameter specifies use of the StartTLS extended operation to establish a TLS session before Binding to the provider. If the **critical** argument is supplied, the session will be aborted if the StartTLS request fails. Otherwise the syncrepl session continues without TLS. The tls_reqcert setting defaults to "demand" and the other TLS settings default to the same as the main slapd TLS settings.

Rather than replicating whole entries, the consumer can query logs of data modifications. This mode of operation is referred to as *delta syncrepl*. In addition to the above parameters, the **logbase** and **logfilter** parameters must be set appropriately for the log that will be used. The **syncdata** parameter must be set to either "accesslog" if the log conforms to the **slapo-accesslog**(5) log format, or "changelog" if the log conforms to the obsolete *changelog* format. If the **syncdata** parameter is omitted or set to "default" then the log parameters are ignored.

updatedn <dn>

This option is only applicable in a slave database. It specifies the DN permitted to update (subject to access controls) the replica. It is only needed in certain push-mode replication scenarios. Generally, this DN *should not* be the same as the **rootdn** used at the master.

updateref <url>

Specify the referral to pass back when **slapd**(8) is asked to modify a replicated local database. If specified multiple times, each url is provided.

DATABASE-SPECIFIC OPTIONS

Each database may allow specific configuration options; they are documented separately in the backends' manual pages. See the **slapd.backends**(5) manual page for an overview of available backends.

EXAMPLES

Here is a short example of a configuration file:

```
include   /etc/openldap/schema/core.schema
pidfile   /var/openldap/run/slapd.pid

# Subtypes of "name" (e.g. "cn" and "ou") with the
# option ";x-hidden" can be searched for/compared,
```

```
# but are not shown.  See slapd.access(5).
attributeoptions x-hidden lang-
access to attrs=name;x-hidden by * =cs

# Protect passwords.  See slapd.access(5).
access    to attrs=userPassword  by * auth
# Read access to other attributes and entries.
access    to *  by * read

database  bdb
suffix    "dc=our-domain,dc=com"
# The database directory MUST exist prior to
# running slapd AND should only be accessible
# by the slapd/tools. Mode 0700 recommended.
directory /var/openldap/openldap-data
# Indices to maintain
index     objectClass  eq
index     cn,sn,mail   pres,eq,approx,sub

# We serve small clients that do not handle referrals,
# so handle remote lookups on their behalf.
database  ldap
suffix    ""
uri       ldap://ldap.some-server.com/
lastmod   off
```

"OpenLDAP Administrator's Guide" contains a longer annotated example of a configuration file. The original /etc/openldap/slapd.conf is another example.

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

SEE ALSO

ldap(3), **slapd−config**(5), **slapd.access**(5), **slapd.backends**(5), **slapd.overlays**(5), **slapd.plugin**(5), **slapd.replog**(5), **slapd**(8), **slapacl**(8), **slapadd**(8), **slapauth**(8), **slapcat**(8), **slapdn**(8), **slapindex**(8), **slappasswd**(8), **slaptest**(8).

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapd.overlays − overlays for slapd, the stand-alone LDAP daemon

DESCRIPTION

The **slapd**(8) daemon can use a variety of different overlays to alter or extend the normal behavior of a database backend. Overlays may be compiled statically into slapd, or when module support is enabled, they may be dynamically loaded. Most of the overlays are only allowed to be configured on individual databases, but some may also be configured globally.

Configuration options for each overlay are documented separately in the corresponding **slapo-<overlay>**(5) manual pages.

accesslog
: Access Logging. This overlay can record accesses to a given backend database on another database.

auditlog
: Audit Logging. This overlay records changes on a given backend database to an LDIF log file. By default it is not built.

chain
: Chaining. This overlay allows automatic referral chasing when a referral would have been returned, either when configured by the server or when requested by the client.

constraint
: Constraint. This overlay enforces a regular expression constraint on all values of specified attributes. It is used to enforce a more rigorous syntax when the underlying attribute syntax is too general.

dds
: Dynamic Directory Services. This overlay supports dynamic objects, which have a limited life after which they expire and are automatically deleted.

dyngroup
: Dynamic Group. This is a demo overlay which extends the Compare operation to detect members of a dynamic group. It has no effect on any other operations.

dynlist
: Dynamic List. This overlay allows expansion of dynamic groups and more.

pcache
: Proxycache. This overlay allows caching of LDAP search requests in a local database. It is most often used with the **slapd-ldap**(5) or **slapd-meta**(5) backends.

ppolicy
: Password Policy. This overlay provides a variety of password control mechanisms, e.g. password aging, password reuse and duplication control, mandatory password resets, etc.

refint
: Referential Integrity. This overlay can be used with a backend database such as **slapd-bdb**(5) to maintain the cohesiveness of a schema which utilizes reference attributes.

retcode
: Return Code. This overlay is useful to test the behavior of clients when server-generated erroneous and/or unusual responses occur.

rwm
: Rewrite/remap. This overlay is experimental. It performs basic DN/data rewrite and objectClass/attributeType mapping.

syncprov
: Syncrepl Provider. This overlay implements the provider-side support for **syncrepl** replication, including persistent search functionality.

translucent
: Translucent Proxy. This overlay can be used with a backend database such as **slapd-bdb**(5) to create a "translucent proxy". Content of entries retrieved from a remote LDAP server can be partially overridden by the database.

unique Attribute Uniqueness. This overlay can be used with a backend database such as **slapd-bdb**(5) to enforce the uniqueness of some or all attributes within a subtree.

valsort Value Sorting. This overlay can be used to enforce a specific order for the values of an attribute when it is returned in a search.

FILES

ETCDIR/slapd.conf
default slapd configuration file

ETCDIR/slapd.d
default slapd configuration directory

SEE ALSO

ldap(3), **slapo−accesslog**(5), **slapo−auditlog**(5), **slapo−chain**(5), **slapo−constraint**(5), **slapo−dds**(5), **slapo−dyngroup**(5), **slapo−dynlist**(5), **slapo−pcache**(5), **slapo−ppolicy**(5), **slapo−refint**(5), **slapo−retcode**(5), **slapo−rwm**(5), **slapo−syncprov**(5), **slapo−translucent**(5), **slapo−unique**(5). **slapo−valsort**(5). **slapd−config**(5), **slapd.conf**(5), **slapd.backends**(5), **slapd**(8). "OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

NAME

slapd.overlays − overlays for slapd, the stand-alone LDAP daemon

DESCRIPTION

The **slapd**(8) daemon can use a variety of different overlays to alter or extend the normal behavior of a database backend. Overlays may be compiled statically into slapd, or when module support is enabled, they may be dynamically loaded. Most of the overlays are only allowed to be configured on individual databases, but some may also be configured globally.

Configuration options for each overlay are documented separately in the corresponding **slapo-<overlay>**(5) manual pages.

accesslog
: Access Logging. This overlay can record accesses to a given backend database on another database.

auditlog
: Audit Logging. This overlay records changes on a given backend database to an LDIF log file. By default it is not built.

chain Chaining. This overlay allows automatic referral chasing when a referral would have been returned, either when configured by the server or when requested by the client.

constraint
: Constraint. This overlay enforces a regular expression constraint on all values of specified attributes. It is used to enforce a more rigorous syntax when the underlying attribute syntax is too general.

dds Dynamic Directory Services. This overlay supports dynamic objects, which have a limited life after which they expire and are automatically deleted.

dyngroup
: Dynamic Group. This is a demo overlay which extends the Compare operation to detect members of a dynamic group. It has no effect on any other operations.

dynlist Dynamic List. This overlay allows expansion of dynamic groups and more.

pcache Proxycache. This overlay allows caching of LDAP search requests in a local database. It is most often used with the **slapd-ldap**(5) or **slapd-meta**(5) backends.

ppolicy
: Password Policy. This overlay provides a variety of password control mechanisms, e.g. password aging, password reuse and duplication control, mandatory password resets, etc.

refint Referential Integrity. This overlay can be used with a backend database such as **slapd-bdb**(5) to maintain the cohesiveness of a schema which utilizes reference attributes.

retcode
: Return Code. This overlay is useful to test the behavior of clients when server-generated erroneous and/or unusual responses occur.

rwm Rewrite/remap. This overlay is experimental. It performs basic DN/data rewrite and objectClass/attributeType mapping.

syncprov
: Syncrepl Provider. This overlay implements the provider-side support for **syncrepl** replication, including persistent search functionality.

translucent
: Translucent Proxy. This overlay can be used with a backend database such as **slapd-bdb**(5) to create a "translucent proxy". Content of entries retrieved from a remote LDAP server can be partially overridden by the database.

unique Attribute Uniqueness. This overlay can be used with a backend database such as **slapd-bdb**(5) to enforce the uniqueness of some or all attributes within a subtree.

valsort Value Sorting. This overlay can be used to enforce a specific order for the values of an attribute when it is returned in a search.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

/etc/openldap/slapd.d
default slapd configuration directory

SEE ALSO

ldap(3), **slapo−accesslog**(5), **slapo−auditlog**(5), **slapo−chain**(5), **slapo−constraint**(5), **slapo−dds**(5), **slapo−dyngroup**(5), **slapo−dynlist**(5), **slapo−pcache**(5), **slapo−ppolicy**(5), **slapo−refint**(5), **slapo−retcode**(5), **slapo−rwm**(5), **slapo−syncprov**(5), **slapo−translucent**(5), **slapo−unique**(5). **slapo−valsort**(5). **slapd−config**(5), **slapd.conf**(5), **slapd.backends**(5), **slapd**(8). "OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapd.plugin − plugin configuration for slapd, the stand-alone LDAP daemon

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **slapd.conf**(5) file contains configuration information for the **slapd**(8) daemon. This configuration file is also used by the SLAPD tools **slapadd**(8), **slapcat**(8), and **slapindex**(8).

The **slapd.conf** file consists of a series of global configuration options that apply to **slapd** as a whole (including all backends), followed by zero or more database backend definitions that contain information specific to a backend instance.

The general format of **slapd.conf** is as follows:

```
# comment - these options apply to every database
<global configuration options>
# first database definition & configuration options
database    <backend 1 type>
<configuration options specific to backend 1>
# subsequent database definitions & configuration options
...
```

If slapd is compiled with *--enable-slapi*, support for plugins according to *Netscape's Directory Server Plug-Ins*. Version 4 of the API is currently implemented, with some extensions from version 5.

Both global and database specific data may contain plugin information. Plugins associated with a specific database are called before global plugins. This manpage details the **slapd**(8) configuration statements that affect the loading of SLAPI *plugins*.

Arguments that should be replaced by actual text are shown in brackets <>.

The structure of the plugin directives is

plugin <type> <lib_path> <init_function> [<arguments>]
Load a plugin of the specified type for the current database.

The **<type>** can be one of **preoperation**, that is executed before processing the operation for the specified database, **postoperation**, that is executed after the operation for the specified database has been processed, **extendedop**, that is used when executing an extended operation, or **object**. The latter is used for miscellaneous types such as ACL, computed attribute and search filter rewriter plugins.

The **<libpath>** argument specifies the path to the plugin loadable object; if a relative path is given, the object is looked for according to the underlying dynamic loading package (libtool's ltdl is used).

The **<init_function>** argument specifies what symbol must be called when the plugin is first loaded. This function should register the functions provided by the plugin for the desired operations. It should be noted that it is this init function, not the plugin type specified as the first argument, that determines when and for what operations the plugin will be invoked. The optional **<arguments>** list is passed to the init function.

pluginlog <file>
Specify an alternative path for the plugin log file (default is LOCALSTATEDIR/errors).

modulepath <pathspec>
This statement sets the module load path for dynamically loadable backends, as described in **slapd.conf**(5); however, since both the dynamically loadable backends and the SLAPI plugins use the same underlying library (libtool's ltdl) its value also affects the plugin search path. In general the search path is made of colon-separated paths; usually the user-defined path is searched first; then the value of the *LTDL_LIBRARY_PATH* environment variable, if defined, is used; finally, the system-specific dynamic load path is attempted (e.g. on Linux the value of the environment variable *LD_LIBRARY_PATH*). Please carefully read the documentation of ltdl because its behavior is very platform dependent.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

LOCALSTATEDIR/errors
: default plugin log file

SEE ALSO

slapd(8),

"OpenLDAP Administrator’s Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

NAME

slapd.plugin − plugin configuration for slapd, the stand-alone LDAP daemon

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **slapd.conf**(5) file contains configuration information for the **slapd**(8) daemon. This configuration file is also used by the SLAPD tools **slapadd**(8), **slapcat**(8), and **slapindex**(8).

The **slapd.conf** file consists of a series of global configuration options that apply to **slapd** as a whole (including all backends), followed by zero or more database backend definitions that contain information specific to a backend instance.

The general format of **slapd.conf** is as follows:

```
# comment - these options apply to every database
<global configuration options>
# first database definition & configuration options
database    <backend 1 type>
<configuration options specific to backend 1>
# subsequent database definitions & configuration options
...
```

If slapd is compiled with *--enable-slapi*, support for plugins according to *Netscape's Directory Server Plug-Ins*. Version 4 of the API is currently implemented, with some extensions from version 5.

Both global and database specific data may contain plugin information. Plugins associated with a specific database are called before global plugins. This manpage details the **slapd**(8) configuration statements that affect the loading of SLAPI *plugins*.

Arguments that should be replaced by actual text are shown in brackets <>.

The structure of the plugin directives is

plugin <type> <lib_path> <init_function> [<arguments>]

Load a plugin of the specified type for the current database.

The **<type>** can be one of **preoperation**, that is executed before processing the operation for the specified database, **postoperation**, that is executed after the operation for the specified database has been processed, **extendedop**, that is used when executing an extended operation, or **object**. The latter is used for miscellaneous types such as ACL, computed attribute and search filter rewriter plugins.

The **<libpath>** argument specifies the path to the plugin loadable object; if a relative path is given, the object is looked for according to the underlying dynamic loading package (libtool's ltdl is used).

The **<init_function>** argument specifies what symbol must be called when the plugin is first loaded. This function should register the functions provided by the plugin for the desired operations. It should be noted that it is this init function, not the plugin type specified as the first argument, that determines when and for what operations the plugin will be invoked. The optional **<arguments>** list is passed to the init function.

pluginlog <file>

Specify an alternative path for the plugin log file (default is /var/openldap/errors).

modulepath <pathspec>

This statement sets the module load path for dynamically loadable backends, as described in **slapd.conf**(5); however, since both the dynamically loadable backends and the SLAPI plugins use the same underlying library (libtool's ltdl) its value also affects the plugin search path. In general the search path is made of colon-separated paths; usually the user-defined path is searched first; then the value of the *LTDL_LIBRARY_PATH* environment variable, if defined, is used; finally, the system-specific dynamic load path is attempted (e.g. on Linux the value of the environment variable *LD_LIBRARY_PATH*). Please carefully read the documentation of ltdl because its behavior is very platform dependent.

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

/var/openldap/errors
: default plugin log file

SEE ALSO

slapd(8),

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapo-accesslog − Access Logging overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Access Logging overlay can be used to record all accesses to a given backend database on another database. This allows all of the activity on a given database to be reviewed using arbitrary LDAP queries, instead of just logging to local flat text files. Configuration options are available for selecting a subset of operation types to log, and to automatically prune older log records from the logging database. Log records are stored with audit schema (see below) to assure their readability whether viewed as LDIF or in raw form.

CONFIGURATION

These **slapd.conf** options apply to the Access Logging overlay. They should appear after the **overlay** directive.

logdb <suffix>

Specify the suffix of a database to be used for storing the log records. The specified database must be defined elsewhere in the configuration. The access controls on the log database should prevent general access. The suffix entry of the log database will be created automatically by this overlay. The log entries will be generated as the immediate children of the suffix entry.

logops <operations>

Specify which types of operations to log. The valid operation types are abandon, add, bind, compare, delete, extended, modify, modrdn, search, and unbind. Aliases for common sets of operations are also available:

writes add, delete, modify, modrdn

reads compare, search

session abandon, bind, unbind

all all operations

logold <filter>

Specify a filter for matching against Deleted and Modified entries. If the entry matches the filter, the old contents of the entry will be logged along with the current request.

logoldattr <attr> ...

Specify a list of attributes whose old contents are always logged in Modify and ModRDN requests. Usually only the contents of attributes that were actually modified will be logged; by default no old attributes are logged for ModRDN requests.

logpurge <age> <interval>

Specify the maximum age for log entries to be retained in the database, and how often to scan the database for old entries. Both the **age** and **interval** are specified as a time span in days, hours, minutes, and seconds. The time format is [ddd+]hh:mm[:ss] i.e., the days and seconds components are optional but hours and minutes are required. Except for days, which can be up to 5 digits, each numeric field must be exactly two digits. For example

logpurge 2+00:00 1+00:00

would specify that the log database should be scanned every day for old entries, and entries older than two days should be deleted. When using a log database that supports ordered indexing on generalizedTime attributes, specifying an eq index on the **reqStart** attribute will greatly benefit the performance of the purge operation.

logsuccess TRUE | FALSE

If set to TRUE then log records will only be generated for successful requests, i.e., requests that produce a result code of 0 (LDAP_SUCCESS). If FALSE, log records are generated for all requests whether they succeed or not. The default is FALSE.

EXAMPLES

```
database bdb
suffix dc=example,dc=com
...
overlay accesslog
logdb cn=log
logops writes reads
logold (objectclass=person)

database bdb
suffix cn=log
...
index reqStart eq
access to *
  by dn.base="cn=admin,dc=example,dc=com" read
```

SCHEMA

The **accesslog** overlay utilizes the "audit" schema described herein. This schema is specifically designed for **accesslog** auditing and is not intended to be used otherwise. It is also noted that the schema described here is *a work in progress*, and hence subject to change without notice. The schema is loaded automatically by the overlay.

The schema includes a number of object classes and associated attribute types as described below.

There is a basic **auditObject** class from which two additional classes, **auditReadObject** and **auditWriteObject** are derived. Object classes for each type of LDAP operation are further derived from these classes. This object class hierarchy is designed to allow flexible yet efficient searches of the log based on either a specific operation type's class, or on more general classifications. The definition of the **auditObject** class is as follows:

```
( 1.3.6.1.4.1.4203.666.11.5.2.1
  NAME 'auditObject'
  DESC 'OpenLDAP request auditing'
  SUP top STRUCTURAL
  MUST ( reqStart $ reqType $ reqSession )
  MAY ( reqDN $ reqAuthzID $ reqControls $ reqRespControls $
    reqEnd $ reqResult $ reqMessage $ reqReferral ) )
```

Note that all of the OIDs used in the logging schema currently reside under the OpenLDAP Experimental branch. It is anticipated that they will migrate to a Standard branch in the future.

An overview of the attributes follows: **reqStart** and **reqEnd** provide the start and end time of the operation, respectively. They use generalizedTime syntax. The **reqStart** attribute is also used as the RDN for each log entry.

The **reqType** attribute is a simple string containing the type of operation being logged, e.g. **add**, **delete**, **search**, etc. For extended operations, the type also includes the OID of the extended operation, e.g. **extended(1.1.1.1)**

The **reqSession** attribute is an implementation-specific identifier that is common to all the operations associated with the same LDAP session. Currently this is slapd's internal connection ID, stored in decimal.

The **reqDN** attribute is the distinguishedName of the target of the operation. E.g., for a Bind request, this is the Bind DN. For an Add request, this is the DN of the entry being added. For a Search request, this is the base DN of the search.

The **reqAuthzID** attribute is the distinguishedName of the user that performed the operation. This will usually be the same name as was established at the start of a session by a Bind request (if any) but may be altered in various circumstances.

The **reqControls** and **reqRespControls** attributes carry any controls sent by the client on the request and returned by the server in the response, respectively. The attribute values are just uninterpreted octet strings.

The **reqResult** attribute is the numeric LDAP result code of the operation, indicating either success or a particular LDAP error code. An error code may be accompanied by a text error message which will be recorded in the **reqMessage** attribute.

The **reqReferral** attribute carries any referrals that were returned with the result of the request.

Operation-specific classes are defined with additional attributes to carry all of the relevant parameters associated with the operation:

```
( 1.3.6.1.4.1.4203.666.11.5.2.4
  NAME 'auditAbandon'
  DESC 'Abandon operation'
  SUP auditObject STRUCTURAL
  MUST reqId )
```

For the **Abandon** operation the **reqId** attribute contains the message ID of the request that was abandoned.

```
( 1.3.6.1.4.1.4203.666.11.5.2.5
  NAME 'auditAdd'
  DESC 'Add operation'
  SUP auditWriteObject STRUCTURAL
  MUST reqMod )
```

The **Add** class inherits from the **auditWriteObject** class. The Add and Modify classes are very similar. The **reqMod** attribute carries all of the attributes of the original entry being added. (Or in the case of a Modify operation, all of the modifications being performed.) The values are formatted as

attribute:<+|-|=|#> [value]

Where '+' indicates an Add of a value, '-' for Delete, '=' for Replace, and '#' for Increment. In an Add operation, all of the reqMod values will have the '+' designator.

```
( 1.3.6.1.4.1.4203.666.11.5.2.6
  NAME 'auditBind'
  DESC 'Bind operation'
  SUP auditObject STRUCTURAL
  MUST ( reqVersion $ reqMethod ) )
```

The **Bind** class includes the **reqVersion** attribute which contains the LDAP protocol version specified in the Bind as well as the **reqMethod** attribute which contains the Bind Method used in the Bind. This will be the string **SIMPLE** for LDAP Simple Binds or **SASL(<mech>)** for SASL Binds. Note that unless configured as a global overlay, only Simple Binds using DNs that reside in the current database will be logged.

```
( 1.3.6.1.4.1.4203.666.11.5.2.7
  NAME 'auditCompare'
  DESC 'Compare operation'
  SUP auditObject STRUCTURAL
  MUST reqAssertion )
```

For the **Compare** operation the **reqAssertion** attribute carries the Attribute Value Assertion used in the compare request.

```
( 1.3.6.1.4.1.4203.666.11.5.2.8
  NAME 'auditDelete'
  DESC 'Delete operation'
  SUP auditWriteObject STRUCTURAL
  MAY reqOld )
```

The **Delete** operation needs no further parameters. However, the **reqOld** attribute may optionally be used to record the contents of the entry prior to its deletion. The values are formatted as

```
attribute: value
```

The **reqOld** attribute is only populated if the entry being deleted matches the configured **logold** filter.

```
( 1.3.6.1.4.1.4203.666.11.5.2.9
  NAME 'auditModify'
  DESC 'Modify operation'
  SUP auditWriteObject STRUCTURAL
  MAY reqOld MUST reqMod )
```

The **Modify** operation contains a description of modifications in the **reqMod** attribute, which was already described above in the Add operation. It may optionally contain the previous contents of any modified attributes in the **reqOld** attribute, using the same format as described above for the Delete operation. The **reqOld** attribute is only populated if the entry being modified matches the configured **logold** filter.

```
( 1.3.6.1.4.1.4203.666.11.5.2.10
  NAME 'auditModRDN'
  DESC 'ModRDN operation'
  SUP auditWriteObject STRUCTURAL
  MUST ( reqNewRDN $ reqDeleteOldRDN )
  MAY ( reqNewSuperior $ reqOld ) )
```

The **ModRDN** class uses the **reqNewRDN** attribute to carry the new RDN of the request. The **reqDeleteOldRDN** attribute is a Boolean value showing **TRUE** if the old RDN was deleted from the entry, or **FALSE** if the old RDN was preserved. The **reqNewSuperior** attribute carries the DN of the new parent entry if the request specified the new parent. The **reqOld** attribute is only populated if the entry being modified matches the configured **logold** filter and contains attributes in the **logoldattr** list.

```
( 1.3.6.1.4.1.4203.666.11.5.2.11
  NAME 'auditSearch'
  DESC 'Search operation'
  SUP auditReadObject STRUCTURAL
  MUST ( reqScope $ reqDerefAliases $ reqAttrsOnly )
  MAY ( reqFilter $ reqAttr $ reqEntries $ reqSizeLimit $
      reqTimeLimit ) )
```

For the **Search** class the **reqScope** attribute contains the scope of the original search request, using the values specified for the LDAP URL format. I.e. **base**, **one**, **sub**, or **subord**. The **reqDerefAliases** attribute is one of **never**, **finding**, **searching**, or **always**, denoting how aliases will be processed during the search. The **reqAttrsOnly** attribute is a Boolean value showing **TRUE** if only attribute names were requested, or **FALSE** if attributes and their values were requested. The **reqFilter** attribute carries the filter used in the search request. The **reqAttr** attribute lists the requested attributes if specific attributes were requested. The **reqEntries** attribute is the integer count of how many entries were returned by this search request. The **reqSizeLimit** and **reqTimeLimit** attributes indicate what limits were requested on the search operation.

```
( 1.3.6.1.4.1.4203.666.11.5.2.12
  NAME 'auditExtended'
  DESC 'Extended operation'
```

SUP auditObject STRUCTURAL
MAY reqData)

The **Extended** class represents an LDAP Extended Operation. As noted above, the actual OID of the operation is included in the **reqType** attribute of the parent class. If any optional data was provided with the request, it will be contained in the **reqData** attribute as an uninterpreted octet string.

NOTES

The Access Log implemented by this overlay may be used for a variety of other tasks, e.g. as a ChangeLog for a replication mechanism, as well as for security/audit logging purposes.

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

This module was written in 2005 by Howard Chu of Symas Corporation.

NAME

slapo-accesslog − Access Logging overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Access Logging overlay can be used to record all accesses to a given backend database on another database. This allows all of the activity on a given database to be reviewed using arbitrary LDAP queries, instead of just logging to local flat text files. Configuration options are available for selecting a subset of operation types to log, and to automatically prune older log records from the logging database. Log records are stored with audit schema (see below) to assure their readability whether viewed as LDIF or in raw form.

CONFIGURATION

These **slapd.conf** options apply to the Access Logging overlay. They should appear after the **overlay** directive.

logdb <suffix>

Specify the suffix of a database to be used for storing the log records. The specified database must be defined elsewhere in the configuration. The access controls on the log database should prevent general access. The suffix entry of the log database will be created automatically by this overlay. The log entries will be generated as the immediate children of the suffix entry.

logops <operations>

Specify which types of operations to log. The valid operation types are abandon, add, bind, compare, delete, extended, modify, modrdn, search, and unbind. Aliases for common sets of operations are also available:

writes add, delete, modify, modrdn

reads compare, search

session abandon, bind, unbind

all all operations

logold <filter>

Specify a filter for matching against Deleted and Modified entries. If the entry matches the filter, the old contents of the entry will be logged along with the current request.

logoldattr <attr> ...

Specify a list of attributes whose old contents are always logged in Modify and ModRDN requests. Usually only the contents of attributes that were actually modified will be logged; by default no old attributes are logged for ModRDN requests.

logpurge <age> <interval>

Specify the maximum age for log entries to be retained in the database, and how often to scan the database for old entries. Both the **age** and **interval** are specified as a time span in days, hours, minutes, and seconds. The time format is [ddd+]hh:mm[:ss] i.e., the days and seconds components are optional but hours and minutes are required. Except for days, which can be up to 5 digits, each numeric field must be exactly two digits. For example

logpurge 2+00:00 1+00:00

would specify that the log database should be scanned every day for old entries, and entries older than two days should be deleted. When using a log database that supports ordered indexing on generalizedTime attributes, specifying an eq index on the **reqStart** attribute will greatly benefit the performance of the purge operation.

logsuccess TRUE | FALSE

If set to TRUE then log records will only be generated for successful requests, i.e., requests that produce a result code of 0 (LDAP_SUCCESS). If FALSE, log records are generated for all requests whether they succeed or not. The default is FALSE.

EXAMPLES

```
database bdb
suffix dc=example,dc=com
...
overlay accesslog
logdb cn=log
logops writes reads
logold (objectclass=person)

database bdb
suffix cn=log
...
index reqStart eq
access to *
  by dn.base="cn=admin,dc=example,dc=com" read
```

SCHEMA

The **accesslog** overlay utilizes the "audit" schema described herein. This schema is specifically designed for **accesslog** auditing and is not intended to be used otherwise. It is also noted that the schema described here is *a work in progress*, and hence subject to change without notice. The schema is loaded automatically by the overlay.

The schema includes a number of object classes and associated attribute types as described below.

There is a basic **auditObject** class from which two additional classes, **auditReadObject** and **auditWriteObject** are derived. Object classes for each type of LDAP operation are further derived from these classes. This object class hierarchy is designed to allow flexible yet efficient searches of the log based on either a specific operation type's class, or on more general classifications. The definition of the **auditObject** class is as follows:

```
( 1.3.6.1.4.1.4203.666.11.5.2.1
  NAME 'auditObject'
  DESC 'OpenLDAP request auditing'
  SUP top STRUCTURAL
  MUST ( reqStart $ reqType $ reqSession )
  MAY ( reqDN $ reqAuthzID $ reqControls $ reqRespControls $
    reqEnd $ reqResult $ reqMessage $ reqReferral ) )
```

Note that all of the OIDs used in the logging schema currently reside under the OpenLDAP Experimental branch. It is anticipated that they will migrate to a Standard branch in the future.

An overview of the attributes follows: **reqStart** and **reqEnd** provide the start and end time of the operation, respectively. They use generalizedTime syntax. The **reqStart** attribute is also used as the RDN for each log entry.

The **reqType** attribute is a simple string containing the type of operation being logged, e.g. **add**, **delete**, **search**, etc. For extended operations, the type also includes the OID of the extended operation, e.g. **extended(1.1.1.1)**

The **reqSession** attribute is an implementation-specific identifier that is common to all the operations associated with the same LDAP session. Currently this is slapd's internal connection ID, stored in decimal.

The **reqDN** attribute is the distinguishedName of the target of the operation. E.g., for a Bind request, this is the Bind DN. For an Add request, this is the DN of the entry being added. For a Search request, this is the base DN of the search.

The **reqAuthzID** attribute is the distinguishedName of the user that performed the operation. This will usually be the same name as was established at the start of a session by a Bind request (if any) but may be altered in various circumstances.

The **reqControls** and **reqRespControls** attributes carry any controls sent by the client on the request and returned by the server in the response, respectively. The attribute values are just uninterpreted octet strings.

The **reqResult** attribute is the numeric LDAP result code of the operation, indicating either success or a particular LDAP error code. An error code may be accompanied by a text error message which will be recorded in the **reqMessage** attribute.

The **reqReferral** attribute carries any referrals that were returned with the result of the request.

Operation-specific classes are defined with additional attributes to carry all of the relevant parameters associated with the operation:

```
( 1.3.6.1.4.1.4203.666.11.5.2.4
  NAME 'auditAbandon'
  DESC 'Abandon operation'
  SUP auditObject STRUCTURAL
  MUST reqId )
```

For the **Abandon** operation the **reqId** attribute contains the message ID of the request that was abandoned.

```
( 1.3.6.1.4.1.4203.666.11.5.2.5
  NAME 'auditAdd'
  DESC 'Add operation'
  SUP auditWriteObject STRUCTURAL
  MUST reqMod )
```

The **Add** class inherits from the **auditWriteObject** class. The Add and Modify classes are very similar. The **reqMod** attribute carries all of the attributes of the original entry being added. (Or in the case of a Modify operation, all of the modifications being performed.) The values are formatted as

attribute:<+|-|=|#> [value]

Where '+' indicates an Add of a value, '-' for Delete, '=' for Replace, and '#' for Increment. In an Add operation, all of the reqMod values will have the '+' designator.

```
( 1.3.6.1.4.1.4203.666.11.5.2.6
  NAME 'auditBind'
  DESC 'Bind operation'
  SUP auditObject STRUCTURAL
  MUST ( reqVersion $ reqMethod ) )
```

The **Bind** class includes the **reqVersion** attribute which contains the LDAP protocol version specified in the Bind as well as the **reqMethod** attribute which contains the Bind Method used in the Bind. This will be the string **SIMPLE** for LDAP Simple Binds or **SASL(<mech>)** for SASL Binds. Note that unless configured as a global overlay, only Simple Binds using DNs that reside in the current database will be logged.

```
( 1.3.6.1.4.1.4203.666.11.5.2.7
  NAME 'auditCompare'
  DESC 'Compare operation'
  SUP auditObject STRUCTURAL
  MUST reqAssertion )
```

For the **Compare** operation the **reqAssertion** attribute carries the Attribute Value Assertion used in the compare request.

```
( 1.3.6.1.4.1.4203.666.11.5.2.8
  NAME 'auditDelete'
  DESC 'Delete operation'
  SUP auditWriteObject STRUCTURAL
  MAY reqOld )
```

The **Delete** operation needs no further parameters. However, the **reqOld** attribute may optionally be used to record the contents of the entry prior to its deletion. The values are formatted as

```
attribute: value
```

The **reqOld** attribute is only populated if the entry being deleted matches the configured **logold** filter.

```
( 1.3.6.1.4.1.4203.666.11.5.2.9
  NAME 'auditModify'
  DESC 'Modify operation'
  SUP auditWriteObject STRUCTURAL
  MAY reqOld MUST reqMod )
```

The **Modify** operation contains a description of modifications in the **reqMod** attribute, which was already described above in the Add operation. It may optionally contain the previous contents of any modified attributes in the **reqOld** attribute, using the same format as described above for the Delete operation. The **reqOld** attribute is only populated if the entry being modified matches the configured **logold** filter.

```
( 1.3.6.1.4.1.4203.666.11.5.2.10
  NAME 'auditModRDN'
  DESC 'ModRDN operation'
  SUP auditWriteObject STRUCTURAL
  MUST ( reqNewRDN $ reqDeleteOldRDN )
  MAY ( reqNewSuperior $ reqOld ) )
```

The **ModRDN** class uses the **reqNewRDN** attribute to carry the new RDN of the request. The **reqDeleteOldRDN** attribute is a Boolean value showing **TRUE** if the old RDN was deleted from the entry, or **FALSE** if the old RDN was preserved. The **reqNewSuperior** attribute carries the DN of the new parent entry if the request specified the new parent. The **reqOld** attribute is only populated if the entry being modified matches the configured **logold** filter and contains attributes in the **logoldattr** list.

```
( 1.3.6.1.4.1.4203.666.11.5.2.11
  NAME 'auditSearch'
  DESC 'Search operation'
  SUP auditReadObject STRUCTURAL
  MUST ( reqScope $ reqDerefAliases $ reqAttrsOnly )
  MAY ( reqFilter $ reqAttr $ reqEntries $ reqSizeLimit $
      reqTimeLimit ) )
```

For the **Search** class the **reqScope** attribute contains the scope of the original search request, using the values specified for the LDAP URL format. I.e. **base**, **one**, **sub**, or **subord**. The **reqDerefAliases** attribute is one of **never**, **finding**, **searching**, or **always**, denoting how aliases will be processed during the search. The **reqAttrsOnly** attribute is a Boolean value showing **TRUE** if only attribute names were requested, or **FALSE** if attributes and their values were requested. The **reqFilter** attribute carries the filter used in the search request. The **reqAttr** attribute lists the requested attributes if specific attributes were requested. The **reqEntries** attribute is the integer count of how many entries were returned by this search request. The **reqSizeLimit** and **reqTimeLimit** attributes indicate what limits were requested on the search operation.

```
( 1.3.6.1.4.1.4203.666.11.5.2.12
  NAME 'auditExtended'
  DESC 'Extended operation'
```

SUP auditObject STRUCTURAL
MAY reqData)

The **Extended** class represents an LDAP Extended Operation. As noted above, the actual OID of the operation is included in the **reqType** attribute of the parent class. If any optional data was provided with the request, it will be contained in the **reqData** attribute as an uninterpreted octet string.

NOTES

The Access Log implemented by this overlay may be used for a variety of other tasks, e.g. as a ChangeLog for a replication mechanism, as well as for security/audit logging purposes.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

This module was written in 2005 by Howard Chu of Symas Corporation.

NAME

slapo-allop − All Operational Attributes overlay

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The All Operational Attributes overlay is designed to allow slapd to interoperate with dumb clients that expect all attributes, including operational ones, to be returned when "*" or an empty attribute list is requested, as opposed to RFC2251 and RFC3673.

CONFIGURATION

These **slapd.conf** options apply to the All Operational overlay. They should appear after the **overlay** directive and before any subsequent **database** directive.

allop-URI <ldapURI>

Specify the base and the scope of search operations that trigger the overlay. By default, it is "ldap:///??base", i.e. it only applies to the rootDSE. This requires the overlay to be instantited as global.

EXAMPLES

default behavior: only affects requests to the rootDSE

```
# global
overlay        allop
```

affects all requests

```
# global
overlay        allop
allop-URI      "ldap:///??sub"
```

affects only requests directed to the suffix of a database

```
# per database
database       bdb
suffix         "dc=example,dc=com"
# database specific directives ...
overlay        allop
allop-URI      "ldap:///dc=example,dc=com??base"
```

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

This module was written in 2005 by Pierangelo Masarati for SysNet s.n.c.

NAME

slapo-auditlog − Audit Logging overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

ETCDIR/slapd.d

DESCRIPTION

The Audit Logging overlay can be used to record all changes on a given backend database to a specified log file. Changes are logged as standard LDIF, with an additional comment header giving the timestamp of the change and the identity of the user making the change.

For Add and Modify operations the identity comes from the modifiersName associated with the operation. This is usually the same as the requestor's identity, but may be set by other overlays to reflect other values.

CONFIGURATION

This **slapd.conf** option applies to the Audit Logging overlay. It should appear after the **overlay** directive.

auditlog <filename>
Specify the fully qualified path for the log file.

olcAuditlogFile <filename>
For use with **cn=config**

EXAMPLE

The following LDIF could be used to add this overlay to **cn=config** (adjust to suit)

```
dn: olcOverlay=auditlog,olcDatabase={1}hdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcAuditLogConfig
olcOverlay: auditlog
olcAuditlogFile: /tmp/auditlog.ldif
```

FILES

ETCDIR/slapd.conf
default slapd configuration file

ETCDIR/slapd.d
default slapd configuration directory

SEE ALSO

slapd.conf(5), **slapd-config(5).**

NAME

slapo-auditlog − Audit Logging overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

/etc/openldap/slapd.d

DESCRIPTION

The Audit Logging overlay can be used to record all changes on a given backend database to a specified log file. Changes are logged as standard LDIF, with an additional comment header giving the timestamp of the change and the identity of the user making the change.

For Add and Modify operations the identity comes from the modifiersName associated with the operation. This is usually the same as the requestor's identity, but may be set by other overlays to reflect other values.

CONFIGURATION

This **slapd.conf** option applies to the Audit Logging overlay. It should appear after the **overlay** directive.

auditlog <filename>
Specify the fully qualified path for the log file.

olcAuditlogFile <filename>
For use with **cn=config**

EXAMPLE

The following LDIF could be used to add this overlay to **cn=config** (adjust to suit)

```
dn: olcOverlay=auditlog,olcDatabase={1}hdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcAuditLogConfig
olcOverlay: auditlog
olcAuditlogFile: /tmp/auditlog.ldif
```

FILES

/etc/openldap/slapd.conf
default slapd configuration file

/etc/openldap/slapd.d
default slapd configuration directory

SEE ALSO

slapd.conf(5), **slapd-config(5).**

NAME

slapo-chain − chain overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **chain** overlay to **slapd**(8) allows automatic referral chasing. Any time a referral is returned (except for bind operations), it is chased by using an instance of the ldap backend. If operations are performed with an identity (i.e. after a bind), that identity can be asserted while chasing the referrals by means of the *identity assertion* feature of back-ldap (see **slapd-ldap**(5) for details), which is essentially based on the **proxied authorization** control [RFC 4370]. Referral chasing can be controlled by the client by issuing the **chaining** control (see *draft-sermersheim-ldap-chaining* for details.)

The config directives that are specific to the **chain** overlay are prefixed by **chain−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

There are very few chain overlay specific directives; however, directives related to the instances of the *ldap* backend that may be implicitly instantiated by the overlay may assume a special meaning when used in conjunction with this overlay. They are described in **slapd-ldap**(5), and they also need to be prefixed by **chain−**.

overlay chain

This directive adds the chain overlay to the current backend. The chain overlay may be used with any backend, but it is mainly intended for use with local storage backends that may return referrals. It is useless in conjunction with the *slapd-ldap* and *slapd-meta* backends because they already exploit the libldap specific referral chase feature. [Note: this may change in the future, as the **ldap**(5) and **meta**(5) backends might no longer chase referrals on their own.]

chain-cache-uri {FALSE|true}

This directive instructs the *chain* overlay to cache connections to URIs parsed out of referrals that are not predefined, to be reused for later chaining. These URIs inherit the properties configured for the underlying **slapd-ldap**(5) before any occurrence of the **chain-uri** directive; basically, they are chained anonymously.

chain-chaining [resolve=<r>] [continuation=<c>] [critical]

This directive enables the *chaining* control (see *draft-sermersheim-ldap-chaining* for details) with the desired resolve and continuation behaviors and criticality. The **resolve** parameter refers to the behavior while discovering a resource, namely when accessing the object indicated by the request DN; the **continuation** parameter refers to the behavior while handling intermediate responses, which is mostly significant for the search operation, but may affect extended operations that return intermediate responses. The values **r** and **c** can be any of **chainingPreferred**, **chainingRequired**, **referralsPreferred**, **referralsRequired**. If the **critical** flag affects the control criticality if provided. [This control is experimental and its support may change in the future.]

chain-max-depth <n>

In case a referral is returned during referral chasing, further chasing occurs at most **<n>** levels deep. Set to **1** (the default) to disable further referral chasing.

chain-return-error {FALSE|true}

In case referral chasing fails, the real error is returned instead of the original referral. In case multiple referral URIs are present, only the first error is returned. This behavior may not be always appropriate nor desirable, since failures in referral chasing might be better resolved by the client (e.g. when caused by distributed authentication issues).

chain-uri <ldapuri>

This directive instantiates a new underlying *ldap* database and instructs it about which URI to contact to chase referrals. As opposed to what stated in **slapd-ldap**(5), only one URI can appear after

this directive; all subsequent **slapd-ldap**(5) directives prefixed by **chain-** refer to this specific instance of a remote server.

Directives for configuring the underlying ldap database may also be required, as shown in this example:

```
overlay             chain
chain-rebind-as-user    FALSE

chain-uri           "ldap://ldap1.example.com"
chain-rebind-as-user    TRUE
chain-idassert-bind     bindmethod="simple"
                        binddn="cn=Auth,dc=example,dc=com"
                        credentials="secret"
                        mode="self"

chain-uri           "ldap://ldap2.example.com"
chain-idassert-bind     bindmethod="simple"
                        binddn="cn=Auth,dc=example,dc=com"
                        credentials="secret"
                        mode="none"
```

Any valid directives for the ldap database may be used; see **slapd-ldap**(5) for details. Multiple occurrences of the **chain-uri** directive may appear, to define multiple "trusted" URIs where operations with *identity assertion* are chained. All URIs not listed in the configuration are chained anonymously. All **slapd-ldap**(5) directives appearing before the first occurrence of **chain-uri** are inherited by all URIs, unless specifically overridden inside each URI configuration.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapd**(8).

AUTHOR

Originally implemented by Howard Chu; extended by Pierangelo Masarati.

NAME

slapo-chain − chain overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **chain** overlay to **slapd**(8) allows automatic referral chasing. Any time a referral is returned (except for bind operations), it is chased by using an instance of the ldap backend. If operations are performed with an identity (i.e. after a bind), that identity can be asserted while chasing the referrals by means of the *identity assertion* feature of back-ldap (see **slapd-ldap**(5) for details), which is essentially based on the **proxied authorization** control [RFC 4370]. Referral chasing can be controlled by the client by issuing the **chaining** control (see *draft-sermersheim-ldap-chaining* for details.)

The config directives that are specific to the **chain** overlay are prefixed by **chain−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

There are very few chain overlay specific directives; however, directives related to the instances of the *ldap* backend that may be implicitly instantiated by the overlay may assume a special meaning when used in conjunction with this overlay. They are described in **slapd-ldap**(5), and they also need to be prefixed by **chain−**.

overlay chain

This directive adds the chain overlay to the current backend. The chain overlay may be used with any backend, but it is mainly intended for use with local storage backends that may return referrals. It is useless in conjunction with the *slapd-ldap* and *slapd-meta* backends because they already exploit the libldap specific referral chase feature. [Note: this may change in the future, as the **ldap**(5) and **meta**(5) backends might no longer chase referrals on their own.]

chain-cache-uri {FALSE|true}

This directive instructs the *chain* overlay to cache connections to URIs parsed out of referrals that are not predefined, to be reused for later chaining. These URIs inherit the properties configured for the underlying **slapd-ldap**(5) before any occurrence of the **chain-uri** directive; basically, they are chained anonymously.

chain-chaining [resolve=<r>] [continuation=<c>] [critical]

This directive enables the *chaining* control (see *draft-sermersheim-ldap-chaining* for details) with the desired resolve and continuation behaviors and criticality. The **resolve** parameter refers to the behavior while discovering a resource, namely when accessing the object indicated by the request DN; the **continuation** parameter refers to the behavior while handling intermediate responses, which is mostly significant for the search operation, but may affect extended operations that return intermediate responses. The values **r** and **c** can be any of **chainingPreferred**, **chainingRequired**, **referralsPreferred**, **referralsRequired**. If the **critical** flag affects the control criticality if provided. [This control is experimental and its support may change in the future.]

chain-max-depth <n>

In case a referral is returned during referral chasing, further chasing occurs at most **<n>** levels deep. Set to **1** (the default) to disable further referral chasing.

chain-return-error {FALSE|true}

In case referral chasing fails, the real error is returned instead of the original referral. In case multiple referral URIs are present, only the first error is returned. This behavior may not be always appropriate nor desirable, since failures in referral chasing might be better resolved by the client (e.g. when caused by distributed authentication issues).

chain-uri <ldapuri>

This directive instantiates a new underlying *ldap* database and instructs it about which URI to contact to chase referrals. As opposed to what stated in **slapd-ldap**(5), only one URI can appear after

this directive; all subsequent **slapd-ldap**(5) directives prefixed by **chain-** refer to this specific instance of a remote server.

Directives for configuring the underlying ldap database may also be required, as shown in this example:

```
overlay             chain
chain-rebind-as-user    FALSE

chain-uri           "ldap://ldap1.example.com"
chain-rebind-as-user    TRUE
chain-idassert-bind     bindmethod="simple"
                        binddn="cn=Auth,dc=example,dc=com"
                        credentials="secret"
                        mode="self"

chain-uri           "ldap://ldap2.example.com"
chain-idassert-bind     bindmethod="simple"
                        binddn="cn=Auth,dc=example,dc=com"
                        credentials="secret"
                        mode="none"
```

Any valid directives for the ldap database may be used; see **slapd-ldap**(5) for details. Multiple occurrences of the **chain-uri** directive may appear, to define multiple "trusted" URIs where operations with *identity assertion* are chained. All URIs not listed in the configuration are chained anonymously. All **slapd-ldap**(5) directives appearing before the first occurrence of **chain-uri** are inherited by all URIs, unless specifically overridden inside each URI configuration.

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapd**(8).

AUTHOR

Originally implemented by Howard Chu; extended by Pierangelo Masarati.

NAME

slapo-constraint − Attribute Constraint Overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The constraint overlay is used to ensure that attribute values match some constraints beyond basic LDAP syntax. Attributes can have multiple constraints placed upon them, and all must be satisfied when modifying an attribute value under constraint.

This overlay is intended to be used to force syntactic regularity upon certain string represented data which have well known canonical forms, like telephone numbers, post codes, FQDNs, etc.

It constrains only LDAP adds and modify commands and only seeks to control the add and modify value of a modify request.

CONFIGURATION

This **slapd.conf** option applies to the constraint overlay. It should appear after the **overlay** directive.

constraint_attribute <attribute_name> <type> <value>

Specifies the constraint which should apply to the attribute named as the first parameter. Two types of constraint are currently supported - **regex** and **uri**.

The parameter following the **regex** type is a Unix style regular expression (See **regex**(7)). The parameter following the **uri** type is an LDAP URI. The URI will be evaluated using an internal search. It must not include a hostname, and it must include a list of attributes to evaluate.

Any attempt to add or modify an attribute named as part of the constraint overlay specification which does not fit the constraint listed will fail with a LDAP_CONSTRAINT_VIOLATION error.

EXAMPLES

```
overlay constraint
constraint_attribute mail regex ^[:alnum:]+@mydomain.com$
constraint_attribute title uri
  ldap:///dc=catalog,dc=example,dc=com?title?sub?(objectClass=titleCatalog)
```

A specification like the above would reject any **mail** attribute which did not look like **<alpha-numeric string>@mydomain.com** It would also reject any **title** attribute whose values were not listed in the **title** attribute of any **titleCatalog** entries in the given scope.

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

This module was written in 2005 by Neil Dunbar of Hewlett-Packard and subsequently extended by Howard Chu and Emmanuel Dreyfus.

NAME

slapo-constraint − Attribute Constraint Overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The constraint overlay is used to ensure that attribute values match some constraints beyond basic LDAP syntax. Attributes can have multiple constraints placed upon them, and all must be satisfied when modifying an attribute value under constraint.

This overlay is intended to be used to force syntactic regularity upon certain string represented data which have well known canonical forms, like telephone numbers, post codes, FQDNs, etc.

It constrains only LDAP adds and modify commands and only seeks to control the add and modify value of a modify request.

CONFIGURATION

This **slapd.conf** option applies to the constraint overlay. It should appear after the **overlay** directive.

constraint_attribute <attribute_name> <type> <value>

Specifies the constraint which should apply to the attribute named as the first parameter. Two types of constraint are currently supported - **regex** and **uri**.

The parameter following the **regex** type is a Unix style regular expression (See **regex**(7)). The parameter following the **uri** type is an LDAP URI. The URI will be evaluated using an internal search. It must not include a hostname, and it must include a list of attributes to evaluate.

Any attempt to add or modify an attribute named as part of the constraint overlay specification which does not fit the constraint listed will fail with a LDAP_CONSTRAINT_VIOLATION error.

EXAMPLES

```
overlay constraint
constraint_attribute mail regex ^[:alnum:]+@mydomain.com$
constraint_attribute title uri
  ldap:///dc=catalog,dc=example,dc=com?title?sub?(objectClass=titleCatalog)
```

A specification like the above would reject any **mail** attribute which did not look like **<alpha-numeric string>@mydomain.com** It would also reject any **title** attribute whose values were not listed in the **title** attribute of any **titleCatalog** entries in the given scope.

FILES

/etc/openldap/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

This module was written in 2005 by Neil Dunbar of Hewlett-Packard and subsequently extended by Howard Chu and Emmanuel Dreyfus. **OpenLDAP Software** is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapo-dds − Dynamic Directory Services overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **dds** overlay to **slapd**(8) implements dynamic objects as per RFC 2589. The name **dds** stands for Dynamic Directory Services. It allows to define dynamic objects, characterized by the **dynamicObject** objectClass.

Dynamic objects have a limited lifetime, determined by a time-to-live (TTL) that can be refreshed by means of a specific **refresh** extended operation. This operation allows to set the Client Refresh Period (CRP), namely the period between refreshes that is required to preserve the dynamic object from expiration. The expiration time is computed by adding the requested TTL to the current time. When dynamic objects reach the end of their lifetime without being further refreshed, they are automatically deleted. There is no guarantee of immediate deletion, so clients should not count on it.

Dynamic objects can have subordinates, provided these also are dynamic objects. RFC 2589 does not specify what the behavior of a dynamic directory service should be when a dynamic object with (dynamic) subordinates expires. In this implementation, the lifetime of dynamic objects with subordinates is prolonged until all the dynamic subordinates expire.

This **slapd.conf**(5) directive adds the **dds** overlay to the current database:

overlay dds

The **dds** overlay may be used with any backend that implements the **add**, **modify**, **search**, and **delete** operations. Since its use may result in many internal entry lookups, adds and deletes, it should be best used in conjunction with backends that have reasonably good write performances.

The config directives that are specific to the **dds** overlay are prefixed by **dds−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

dds−max−ttl <ttl>
: Specifies the max TTL value. This is also the default TTL newly created dynamic objects receive, unless **dds−default−ttl** is set. When the client with a refresh extended operation requests a TTL higher than it, sizeLimitExceeded is returned. This value must be between 86400 (1 day, the default) and 31557600 (1 year plus 6 hours, as per RFC 2589).

dds−min−ttl <ttl>
: Specifies the min TTL value; clients requesting a lower TTL by means of the refresh extended operation actually obtain this value as CRP. If set to 0 (the default), no lower limit is set.

dds−default−ttl <ttl>
: Specifies the default TTL value that newly created dynamic objects get. If set to 0 (the default), the **dds−max−ttl** is used.

dds−interval <ttl>
: Specifies the interval between expiration checks; defaults to 1 hour.

dds−tolerance <ttl>
Specifies an extra time that is added to the timer that actually wakes up the thread that will delete an expired dynamic object. So the nominal lifetime of the entry is that specified in the **entryTtl** attribute, but its lifetime will actually be **entryTtl + tolerance**. Note that there is no guarantee that the lifetime of a dynamic object will be *exactly* the requested TTL; due to implementation details, it may be longer, which is allowed by RFC 2589. By default, tolerance is 0.

dds−max−dynamicObjects <num>
Specifies the maximum number of dynamic objects that can simultaneously exist within a naming context. This allows to limit the amount of resources (mostly in terms of run-queue size) that are used by dynamic objects. By default, no limit is set.

dds-state {TRUE|false}
Specifies if the Dynamic Directory Services feature is enabled or not. By default it is; however, a proxy does not need to keep track of dynamic objects itself, it only needs to inform the frontend that support for dynamic objects is available.

ACCESS CONTROL

The **dds** overlay restricts the refresh operation by requiring **manage** access to the **entryTtl** attribute (see **slapd.access**(5) for details about the **manage** access privilege). Since the **entryTtl** is an operational, NO-USER-MODIFICATION attribute, no direct write access to it is possible. So the **dds** overlay turns refresh extended operation into an internal modification to the value of the **entryTtl** attribute with the **manageDIT** control set.

RFC 2589 recommends that anonymous clients should not be allowed to refresh a dynamic object. This can be implemented by appropriately crafting access control to obtain the desired effect.

Example: restrict refresh to authenticated clients

```
access to attrs=entryTtl
        by users manage
        by * read
```

Example: restrict refresh to the creator of the dynamic object

```
access to attrs=entryTtl
        by dnattr=creatorsName manage
        by * read
```

Another suggested usage of dynamic objects is to implement dynamic meetings; in this case, all the participants to the meeting are allowed to refresh the meeting object, but only the creator can delete it (otherwise it will be deleted when the TTL expires)

Example: assuming *participant* is a valid DN-valued attribute, allow users to start a meeting and to join it; restrict refresh to the participants; restrict delete to the creator

```
access to dn.base="cn=Meetings"
                attrs=children
        by users write

access to dn.onelevel="cn=Meetings"
                attrs=entry
        by dnattr=creatorsName write
```

```
        by * read

access to dn.onelevel="cn=Meetings"
        attrs=participant
        by dnattr=creatorsName write
        by users selfwrite
        by * read

access to dn.onelevel="cn=Meetings"
        attrs=entryTtl
        by dnattr=participant manage
        by * read
```

REPLICATION

This implementation of RFC 2589 provides a restricted interpretation of how dynamic objects replicate. Only the master takes care of handling dynamic object expiration, while replicas simply see the dynamic object as a plain object.

When replicating these objects, one needs to explicitly exclude the **dynamicObject** class and the **entryTtl** attribute. This implementation of RFC 2589 introduces a new operational attribute, **entryExpireTimestamp**, that contains the expiration timestamp. This must be excluded from replication as well.

The quick and dirty solution is to set **schemacheck=off** in the syncrepl configuration and, optionally, exclude the operational attributes from replication, using

```
syncrepl ...
        exattrs=entryTtl,entryExpireTimestamp
```

In any case the overlay must be either statically built in or run-time loaded by the consumer, so that it is aware of the **entryExpireTimestamp** operational attribute; however, it must not be configured in the shadow database. Currently, there is no means to remove the **dynamicObject** class from the entry; this may be seen as a feature, since it allows to see the dynamic properties of the object.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8).

AUTHOR

Implemented by Pierangelo Masarati.

NAME

slapo-dds − Dynamic Directory Services overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **dds** overlay to **slapd**(8) implements dynamic objects as per RFC 2589. The name **dds** stands for Dynamic Directory Services. It allows to define dynamic objects, characterized by the **dynamicObject** objectClass.

Dynamic objects have a limited lifetime, determined by a time-to-live (TTL) that can be refreshed by means of a specific **refresh** extended operation. This operation allows to set the Client Refresh Period (CRP), namely the period between refreshes that is required to preserve the dynamic object from expiration. The expiration time is computed by adding the requested TTL to the current time. When dynamic objects reach the end of their lifetime without being further refreshed, they are automatically deleted. There is no guarantee of immediate deletion, so clients should not count on it.

Dynamic objects can have subordinates, provided these also are dynamic objects. RFC 2589 does not specify what the behavior of a dynamic directory service should be when a dynamic object with (dynamic) subordinates expires. In this implementation, the lifetime of dynamic objects with subordinates is prolonged until all the dynamic subordinates expire.

This **slapd.conf**(5) directive adds the **dds** overlay to the current database:

overlay dds

The **dds** overlay may be used with any backend that implements the **add**, **modify**, **search**, and **delete** operations. Since its use may result in many internal entry lookups, adds and deletes, it should be best used in conjunction with backends that have reasonably good write performances.

The config directives that are specific to the **dds** overlay are prefixed by **dds−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

dds−max−ttl <ttl>

Specifies the max TTL value. This is also the default TTL newly created dynamic objects receive, unless **dds−default−ttl** is set. When the client with a refresh extended operation requests a TTL higher than it, sizeLimitExceeded is returned. This value must be between 86400 (1 day, the default) and 31557600 (1 year plus 6 hours, as per RFC 2589).

dds−min−ttl <ttl>

Specifies the min TTL value; clients requesting a lower TTL by means of the refresh extended operation actually obtain this value as CRP. If set to 0 (the default), no lower limit is set.

dds−default−ttl <ttl>

Specifies the default TTL value that newly created dynamic objects get. If set to 0 (the default), the **dds−max−ttl** is used.

dds−interval <ttl>

Specifies the interval between expiration checks; defaults to 1 hour.

dds−tolerance <ttl>

Specifies an extra time that is added to the timer that actually wakes up the thread that will delete an expired dynamic object. So the nominal lifetime of the entry is that specified in the **entryTtl** attribute, but its lifetime will actually be **entryTtl + tolerance**. Note that there is no guarantee that the lifetime of a dynamic object will be *exactly* the requested TTL; due to implementation details, it may be longer, which is allowed by RFC 2589. By default, tolerance is 0.

dds−max−dynamicObjects <num>

Specifies the maximum number of dynamic objects that can simultaneously exist within a naming context. This allows to limit the amount of resources (mostly in terms of run-queue size) that are used by dynamic objects. By default, no limit is set.

dds-state {TRUE|false}

Specifies if the Dynamic Directory Services feature is enabled or not. By default it is; however, a proxy does not need to keep track of dynamic objects itself, it only needs to inform the frontend that support for dynamic objects is available.

ACCESS CONTROL

The **dds** overlay restricts the refresh operation by requiring **manage** access to the **entryTtl** attribute (see **slapd.access**(5) for details about the **manage** access privilege). Since the **entryTtl** is an operational, NO-USER-MODIFICATION attribute, no direct write access to it is possible. So the **dds** overlay turns refresh extended operation into an internal modification to the value of the **entryTtl** attribute with the **manageDIT** control set.

RFC 2589 recommends that anonymous clients should not be allowed to refresh a dynamic object. This can be implemented by appropriately crafting access control to obtain the desired effect.

Example: restrict refresh to authenticated clients

```
access to attrs=entryTtl
        by users manage
        by * read
```

Example: restrict refresh to the creator of the dynamic object

```
access to attrs=entryTtl
        by dnattr=creatorsName manage
        by * read
```

Another suggested usage of dynamic objects is to implement dynamic meetings; in this case, all the participants to the meeting are allowed to refresh the meeting object, but only the creator can delete it (otherwise it will be deleted when the TTL expires)

Example: assuming *participant* is a valid DN-valued attribute, allow users to start a meeting and to join it; restrict refresh to the participants; restrict delete to the creator

```
access to dn.base="cn=Meetings"
                attrs=children
        by users write

access to dn.onelevel="cn=Meetings"
                attrs=entry
        by dnattr=creatorsName write
```

```
        by * read

access to dn.onelevel="cn=Meetings"
        attrs=participant
        by dnattr=creatorsName write
        by users selfwrite
        by * read

access to dn.onelevel="cn=Meetings"
        attrs=entryTtl
        by dnattr=participant manage
        by * read
```

REPLICATION

This implementation of RFC 2589 provides a restricted interpretation of how dynamic objects replicate. Only the master takes care of handling dynamic object expiration, while replicas simply see the dynamic object as a plain object.

When replicating these objects, one needs to explicitly exclude the **dynamicObject** class and the **entryTtl** attribute. This implementation of RFC 2589 introduces a new operational attribute, **entryExpireTimestamp**, that contains the expiration timestamp. This must be excluded from replication as well.

The quick and dirty solution is to set **schemacheck=off** in the syncrepl configuration and, optionally, exclude the operational attributes from replication, using

```
syncrepl ...
        exattrs=entryTtl,entryExpireTimestamp
```

In any case the overlay must be either statically built in or run-time loaded by the consumer, so that it is aware of the **entryExpireTimestamp** operational attribute; however, it must not be configured in the shadow database. Currently, there is no means to remove the **dynamicObject** class from the entry; this may be seen as a feature, since it allows to see the dynamic properties of the object.

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8).

AUTHOR

Implemented by Pierangelo Masarati.

NAME

slapo-dyngroup − Dynamic Group overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Dynamic Group overlay allows clients to use LDAP Compare operations to test the membership of a dynamic group the same way they would check against a static group. Compare operations targeting a group's static member attribute will be intercepted and tested against the configured dynamic group's URL attribute.

Note that this intercept only happens if the actual Compare operation does not return a LDAP_COMPARE_TRUE result. So if a group has both static and dynamic members, the static member list will be checked first.

CONFIGURATION

This **slapd.conf** option applies to the Dynamic Group overlay. It should appear after the **overlay** directive.

attrpair <memberAttr> <URLattr>

Specify the attributes to be compared. A compare operation on the *memberAttr* will cause the *URLattr* to be evaluated for the result.

EXAMPLES

```
database bdb
...
overlay dyngroup
attrpair member memberURL
```

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5).

AUTHOR

Howard Chu

NAME

slapo-dyngroup − Dynamic Group overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Dynamic Group overlay allows clients to use LDAP Compare operations to test the membership of a dynamic group the same way they would check against a static group. Compare operations targeting a group's static member attribute will be intercepted and tested against the configured dynamic group's URL attribute.

Note that this intercept only happens if the actual Compare operation does not return a LDAP_COMPARE_TRUE result. So if a group has both static and dynamic members, the static member list will be checked first.

CONFIGURATION

This **slapd.conf** option applies to the Dynamic Group overlay. It should appear after the **overlay** directive.

attrpair <memberAttr> <URLattr>

Specify the attributes to be compared. A compare operation on the *memberAttr* will cause the *URLattr* to be evaluated for the result.

EXAMPLES

```
database bdb
...
overlay dyngroup
attrpair member memberURL
```

FILES

/etc/openldap/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5).

AUTHOR

Howard Chu

NAME

slapo-dynlist − Dynamic List overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **dynlist** overlay to **slapd**(8) allows expansion of dynamic groups and more. Any time an entry with a specific objectClass is being returned, the LDAP URI-valued occurrences of a specific attribute are expanded into the corresponding entries, and the values of the attributes listed in the URI are added to the original entry. No recursion is allowed, to avoid potential infinite loops. The resulting entry must comply with the LDAP data model, so constraints are enforced. For example, if a *SINGLE-VALUE* attribute is listed, only the first value results in the final entry. The above described behavior is disabled when the *manageDSAit* control (RFC 3296) is used. In that case, the contents of the dynamic group entry is returned; namely, the URLs are returned instead of being expanded.

CONFIGURATION

The config directives that are specific to the **dynlist** overlay must be prefixed by **dynlist−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

overlay dynlist

This directive adds the dynlist overlay to the current database, or to the frontend, if used before any database instantiation; see **slapd.conf**(5) for details.

This **slapd.conf** configuration option is defined for the dynlist overlay. It may have multiple occurrences, and it must appear after the **overlay** directive.

dynlist-attrset <group-oc> <URL-ad> [[<mapped-ad>:]<member-ad> ...]

The value **<group-oc>** is the name of the objectClass that triggers the dynamic expansion of the data.

The value **<URL-ad>** is the name of the attributeDescription that contains the URI that is expanded by the overlay; if none is present, no expansion occurs. If the intersection of the attributes requested by the search operation (or the asserted attribute for compares) and the attributes listed in the URI is empty, no expansion occurs for that specific URI. It must be a subtype of *labeledURI*.

The value **<member-ad>** is optional; if present, the overlay behaves as a dynamic group: this attribute will list the DN of the entries resulting from the internal search. In this case, the <attrs> portion of the URI must be absent, and the DNs of all the entries resulting from the expansion of the URI are listed as values of this attribute. Compares that assert the value of the **<member-ad>** attribute of entries with **<group-oc>** objectClass apply as if the DN of the entries resulting from the expansion of the URI were present in the **<group-oc>** entry as values of the **<member-ad>** attribute.

Alternatively, **<mapped-ad>:<member-ad>** can be used to remap attributes obtained through expansion. **<member-ad>** attributes are not filled by expanded DN, but are remapped as **<mapped-ad>** attributes. Multiple mapping statements can be used.

The dynlist overlay may be used with any backend, but it is mainly intended for use with local storage backends. In case the URI expansion is very resource-intensive and occurs frequently with well-defined patterns, one should consider adding a proxycache later on in the overlay stack.

AUTHORIZATION

By default the expansions are performed using the identity of the current LDAP user. This identity may be overridden by setting the **dgIdentity** attribute to the DN of another LDAP user. In that case the dgIdentity will be used when expanding the URIs in the object. Setting the dgIdentity to a zero-length string will cause the expansions to be performed anonymously. Note that the dgIdentity attribute is defined in the **dyngroup** schema, and this schema must be loaded before the dgIdentity authorization feature may be used.

EXAMPLE

This example collects all the email addresses of a database into a single entry; first of all, make sure that slapd.conf contains the directives:

```
include /path/to/dyngroup.schema
# ...

database <database>
# ...

overlay dynlist
dynlist-attrset groupOfURLs memberURL
```

and that slapd loads dynlist.la, if compiled as a run-time module; then add to the database an entry like

```
dn: cn=Dynamic List,ou=Groups,dc=example,dc=com
objectClass: groupOfURLs
cn: Dynamic List
memberURL: ldap:///ou=People,dc=example,dc=com?mail?sub?(objectClass=person)
```

If no <attrs> are provided in the URI, all (non-operational) attributes are collected.

This example implements the dynamic group feature on the **member** attribute:

```
include /path/to/dyngroup.schema
# ...

database <database>
# ...

overlay dynlist
dynlist-attrset groupOfURLs memberURL member
```

A dynamic group with dgIdentity authorization could be created with an entry like

```
dn: cn=Dynamic Group,ou=Groups,dc=example,dc=com
objectClass: groupOfURLs
objectClass: dgIdentityAux
cn: Dynamic Group
memberURL: ldap:///ou=People,dc=example,dc=com??sub?(objectClass=person)
dgIdentity: cn=Group Proxy,ou=Services,dc=example,dc=com
```

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8). The **slapo-dynlist**(5) overlay supports dynamic configuration via **back-config**.

ACKNOWLEDGEMENTS

This module was written in 2004 by Pierangelo Masarati for SysNet s.n.c.

Attribute remapping was contributed in 2008 by Emmanuel Dreyfus.

NAME

slapo-dynlist − Dynamic List overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **dynlist** overlay to **slapd**(8) allows expansion of dynamic groups and more. Any time an entry with a specific objectClass is being returned, the LDAP URI-valued occurrences of a specific attribute are expanded into the corresponding entries, and the values of the attributes listed in the URI are added to the original entry. No recursion is allowed, to avoid potential infinite loops. The resulting entry must comply with the LDAP data model, so constraints are enforced. For example, if a *SINGLE-VALUE* attribute is listed, only the first value results in the final entry. The above described behavior is disabled when the *manageDSAit* control (RFC 3296) is used. In that case, the contents of the dynamic group entry is returned; namely, the URLs are returned instead of being expanded.

CONFIGURATION

The config directives that are specific to the **dynlist** overlay must be prefixed by **dynlist−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

overlay dynlist

This directive adds the dynlist overlay to the current database, or to the frontend, if used before any database instantiation; see **slapd.conf**(5) for details.

This **slapd.conf** configuration option is defined for the dynlist overlay. It may have multiple occurrences, and it must appear after the **overlay** directive.

dynlist-attrset <group-oc> <URL-ad> [[<mapped-ad>:]<member-ad> ...]

The value **<group-oc>** is the name of the objectClass that triggers the dynamic expansion of the data.

The value **<URL-ad>** is the name of the attributeDescription that contains the URI that is expanded by the overlay; if none is present, no expansion occurs. If the intersection of the attributes requested by the search operation (or the asserted attribute for compares) and the attributes listed in the URI is empty, no expansion occurs for that specific URI. It must be a subtype of *labeledURI*.

The value **<member-ad>** is optional; if present, the overlay behaves as a dynamic group: this attribute will list the DN of the entries resulting from the internal search. In this case, the <attrs> portion of the URI must be absent, and the DNs of all the entries resulting from the expansion of the URI are listed as values of this attribute. Compares that assert the value of the **<member-ad>** attribute of entries with **<group-oc>** objectClass apply as if the DN of the entries resulting from the expansion of the URI were present in the **<group-oc>** entry as values of the **<member-ad>** attribute.

Alternatively, **<mapped-ad>:<member-ad>** can be used to remap attributes obtained through expansion. **<member-ad>** attributes are not filled by expanded DN, but are remapped as **<mapped-ad>** attributes. Multiple mapping statements can be used.

The dynlist overlay may be used with any backend, but it is mainly intended for use with local storage backends. In case the URI expansion is very resource-intensive and occurs frequently with well-defined patterns, one should consider adding a proxycache later on in the overlay stack.

AUTHORIZATION

By default the expansions are performed using the identity of the current LDAP user. This identity may be overridden by setting the **dgIdentity** attribute to the DN of another LDAP user. In that case the dgIdentity will be used when expanding the URIs in the object. Setting the dgIdentity to a zero-length string will cause the expansions to be performed anonymously. Note that the dgIdentity attribute is defined in the **dyngroup** schema, and this schema must be loaded before the dgIdentity authorization feature may be used.

EXAMPLE

This example collects all the email addresses of a database into a single entry; first of all, make sure that slapd.conf contains the directives:

```
include /path/to/dyngroup.schema
# ...

database <database>
# ...

overlay dynlist
dynlist-attrset groupOfURLs memberURL
```

and that slapd loads dynlist.la, if compiled as a run-time module; then add to the database an entry like

```
dn: cn=Dynamic List,ou=Groups,dc=example,dc=com
objectClass: groupOfURLs
cn: Dynamic List
memberURL: ldap:///ou=People,dc=example,dc=com?mail?sub?(objectClass=person)
```

If no <attrs> are provided in the URI, all (non-operational) attributes are collected.

This example implements the dynamic group feature on the **member** attribute:

```
include /path/to/dyngroup.schema
# ...

database <database>
# ...

overlay dynlist
dynlist-attrset groupOfURLs memberURL member
```

A dynamic group with dgIdentity authorization could be created with an entry like

```
dn: cn=Dynamic Group,ou=Groups,dc=example,dc=com
objectClass: groupOfURLs
objectClass: dgIdentityAux
cn: Dynamic Group
memberURL: ldap:///ou=People,dc=example,dc=com??sub?(objectClass=person)
dgIdentity: cn=Group Proxy,ou=Services,dc=example,dc=com
```

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8). The **slapo-dynlist**(5) overlay supports dynamic configuration via **back-config**.

ACKNOWLEDGEMENTS

This module was written in 2004 by Pierangelo Masarati for SysNet s.n.c.

Attribute remapping was contributed in 2008 by Emmanuel Dreyfus.

NAME

slapo-lastmod − Last Modification overlay

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **lastmod** overlay creates a service entry rooted at the suffix of the database it's stacked onto, which holds the DN, the modification type, the modifiersName and the modifyTimestamp of the last write operation performed on that database. The lastmod overlay cannot be used when the "lastmod" feature is disabled, i.e. "lastmod off" is used.

All operations targeted at the DN of the lastmod entry are rejected, except reads, i.e. searches with **base** scope. Regular operations are ignored, unless they result in writing; then, in case of success, the lastmod entry is updated accordingly, if possible.

CONFIGURATION

These **slapd.conf** configuration options apply to the lastmod overlay. They must appear after the **overlay** directive.

lastmod-rdnvalue <RDN value>

Specify the value of the RDN used for the service entry. By default *Lastmod* is used.

lastmod-enabled {yes|no}

Specify whether the overlay must be enabled or not at startup. By default, the overlay is enabled; however, by changing the boolean value of the attribute *lastmodEnabled*, one can affect the status of the overlay. This is useful, for instance, to inhibit the overlay from keeping track of large bulk loads or deletions.

OBJECT CLASS

The **lastmod** overlay depends on the **lastmod** objectClass. The definition of that class is as follows:

```
( 1.3.6.1.4.1.4203.666.3.13 "
  NAME 'lastmod'
  DESC 'OpenLDAP per-database last modification monitoring'
  STRUCTURAL
  SUP top
  MUST ( cn $ lastmodDN $ lastmodType )
  MAY ( description $ seeAlso ) )
```

ATTRIBUTES

Each one of the sections below details the meaning and use of a particular attribute of this **lastmod** objectClass. Most of the attributes that are specific to the lastmod objectClass are operational, since they can logically be altered only by the DSA. The most notable exception is the *lastmodEnabled* attributeType, which can be altered via protocol to change the status of the overlay.

lastmodEnabled

This attribute contains a boolean flag that determines the status of the overlay. It can be altered via protocol by issuing a modify operation that replaces the value of the attribute.

```
(  1.3.6.1.4.1.4203.666.1.30
  NAME 'lastmodEnabled'
  DESC 'Lastmod overlay state'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  EQUALITY booleanMatch
  SINGLE-VALUE )
```

OPERATIONAL ATTRIBUTES

Each one of the sections below details the meaning and use of a particular attribute of this **lastmod** objectClass. Most of the attributes that are specific to the lastmod objectClass are operational, since they can logically be altered only by the DSA.

lastmodDN

This attribute contains the distinguished name of the entry that was last modified within the naming context of a database.

```
( 1.3.6.1.4.1.4203.666.1.28
  NAME 'lastmodDN'
  DESC 'DN of last modification'
  EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  NO-USER-MODIFICATION
  USAGE directoryOperation )
```

lastmodType

This attribute contains the type of the modification that occurred to the last modified entry. Legal values are **add**, **delete**, **exop**, **modify**, **modrdn** and **unknown**. The latter should only be used as a fall-thru in case of unhandled request types that are considered equivalent to a write operation.

```
( 1.3.6.1.4.1.4203.666.1.29
  NAME 'lastmodType'
  DESC 'Type of last modification'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  EQUALITY caseIgnoreMatch
  SINGLE-VALUE
  NO-USER-MODIFICATION
  USAGE directoryOperation )
```

EXAMPLES

```
database    bdb
suffix      dc=example,dc=com
overlay     lastmod
lastmod-rdnvalue "Last Modification"
```

SEE ALSO

ldap(3), **slapd.conf**(5),

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

BUGS

It is unclear whether this overlay can safely interoperate with other overlays. If the underlying backend does not implement entry_get/entry_release handlers, modrdn update can become tricky. The code needs some cleanup and more consistent error handling. So far, the OIDs for the schema haven't been assigned yet.

ACKNOWLEDGEMENTS

This module was written in 2004 by Pierangelo Masarati in fulfillment of requirements from SysNet s.n.c.; this man page has been copied from **slapo-ppolicy**(5), and most of the overlays ever written are copied from Howard Chu's first overlays.

OpenLDAP is developed and maintained by The OpenLDAP Project (http://www.openldap.org/). **OpenLDAP** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapo-memberof − Reverse Group Membership overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **memberof** overlay to **slapd**(8) allows automatic reverse group membership maintenance. Any time a group entry is modified, its members are modified as appropriate in order to keep a DN-valued "is member of" attribute updated with the DN of the group.

CONFIGURATION

The config directives that are specific to the **memberof** overlay must be prefixed by **memberof−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

overlay memberof

This directive adds the memberof overlay to the current database; see **slapd.conf**(5) for details.

The following **slapd.conf** configuration options are defined for the memberofoverlay.

memberof-group-oc <*group-oc*>

The value <*group-oc*> is the name of the objectClass that triggers the reverse group membership update. It defaults to *groupOfNames*.

memberof-member-ad <*member-ad*>

The value <*member-ad*> is the name of the attribute that contains the names of the members in the group objects; it must be DN-valued. It defaults to *member*.

memberof-memberof-ad <*memberof-ad*>

The value <*memberof-ad*> is the name of the attribute that contains the names of the groups an entry is member of; it must be DN-valued. Its contents are automatically updated by the overlay. It defaults to *memberOf*.

memberof-dn <*dn*>

The value <*dn*> contains the DN that is used as *modifiersName* for internal modifications performed to update the reverse group membership. It defaults to the *rootdn* of the underlying database.

memberof-dangling {*ignore***,** *drop***,** *error***}**

This option determines the behavior of the overlay when, during a modification, it encounters dangling references. The default is *ignore*, which may leave dangling references. Other options are *drop*, which discards those modifications that would result in dangling references, and *error*, which causes modifications that would result in dangling references to fail.

memberof-dangling-error <*error-code*>

If **memberof-dangling** is set to *error*, this configuration parameter can be used to modify the response code returned in case of violation. It defaults to "constraint violation", but other implementations are known to return "no such object" instead.

memberof-refint {*true***|***FALSE***}**

This option determines whether the overlay will try to preserve referential integrity or not. If set to *TRUE*, when an entry containing values of the "is member of" attribute is modified, the

corresponding groups are modified as well.

The memberof overlay may be used with any backend that provides full read-write functionality, but it is mainly intended for use with local storage backends.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8). The **slapo-memberof**(5) overlay supports dynamic configuration via **back-config**.

ACKNOWLEDGEMENTS

This module was written in 2005 by Pierangelo Masarati for SysNet s.n.c.

NAME

slapo-memberof − Reverse Group Membership overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **memberof** overlay to **slapd**(8) allows automatic reverse group membership maintenance. Any time a group entry is modified, its members are modified as appropriate in order to keep a DN-valued "is member of" attribute updated with the DN of the group.

CONFIGURATION

The config directives that are specific to the **memberof** overlay must be prefixed by **memberof−**, to avoid potential conflicts with directives specific to the underlying database or to other stacked overlays.

overlay memberof

This directive adds the memberof overlay to the current database; see **slapd.conf**(5) for details.

The following **slapd.conf** configuration options are defined for the memberofoverlay.

memberof-group-oc <*group-oc*>

The value <*group-oc*> is the name of the objectClass that triggers the reverse group membership update. It defaults to *groupOfNames*.

memberof-member-ad <*member-ad*>

The value <*member-ad*> is the name of the attribute that contains the names of the members in the group objects; it must be DN-valued. It defaults to *member*.

memberof-memberof-ad <*memberof-ad*>

The value <*memberof-ad*> is the name of the attribute that contains the names of the groups an entry is member of; it must be DN-valued. Its contents are automatically updated by the overlay. It defaults to *memberOf*.

memberof-dn <*dn*>

The value <*dn*> contains the DN that is used as *modifiersName* for internal modifications performed to update the reverse group membership. It defaults to the *rootdn* of the underlying database.

memberof-dangling {*ignore***,** *drop***,** *error***}**

This option determines the behavior of the overlay when, during a modification, it encounters dangling references. The default is *ignore*, which may leave dangling references. Other options are *drop*, which discards those modifications that would result in dangling references, and *error*, which causes modifications that would result in dangling references to fail.

memberof-dangling-error <*error-code*>

If **memberof-dangling** is set to *error*, this configuration parameter can be used to modify the response code returned in case of violation. It defaults to "constraint violation", but other implementations are known to return "no such object" instead.

memberof-refint {*true***|***FALSE***}**

This option determines whether the overlay will try to preserve referential integrity or not. If set to *TRUE*, when an entry containing values of the "is member of" attribute is modified, the

corresponding groups are modified as well.

The memberof overlay may be used with any backend that provides full read-write functionality, but it is mainly intended for use with local storage backends.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8). The **slapo-memberof**(5) overlay supports dynamic configuration via **back-config**.

ACKNOWLEDGEMENTS

This module was written in 2005 by Pierangelo Masarati for SysNet s.n.c.

NAME

slapo-pcache − proxycache overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **pcache** overlay to **slapd**(8) allows caching of LDAP search requests (queries) in a local database. For an incoming query, the proxy cache determines its corresponding **template**. If the template was specified as cacheable using the **proxytemplate** directive and the request is contained in a cached request, it is answered from the proxy cache. Otherwise, the search is performed as usual and cacheable search results are saved in the cache for use in future queries.

A template is defined by a filter string and an index identifying a set of attributes. The **template string** for a query can be obtained by removing assertion values from the RFC 4515 representation of its search filter. A query belongs to a template if its template string and set of projected attributes correspond to a cacheable template. Examples of template strings are **(mail=)**, **(|(sn=)(cn=))**, **(&(sn=)(givenName=))**.

The config directives that are specific to the **proxycache** overlay can be prefixed by **proxycache−**, to avoid conflicts with directives specific to the underlying database or to other stacked overlays. This may be particularly useful for those directives that refer to the backend used for local storage. The following cache specific directives can be used to configure the proxy cache:

overlay pcache

This directive adds the proxy cache overlay to the current backend. The proxy cache overlay may be used with any backend but is intended for use with the **ldap**, **meta**, and **sql** backends.

proxycache <database> <max_entries> <numattrsets> <entry_limit> <cc_period>

The directive enables proxy caching in the current backend and sets general cache parameters. A <database> backend will be used internally to maintain the cached entries. The chosen database will need to be configured as well, as shown below. Cache replacement is invoked when the cache size grows to <max_entries> entries and continues till the cache size drops below this size. <numattrsets> should be equal to the number of following **proxyattrset** directives. Queries are cached only if they correspond to a cacheable template (specified by the **proxytemplate** directive) and the number of entries returned is less than <entry_limit>. Consistency check is performed every <cc_period> duration (specified in secs). In each cycle queries with expired "time to live(**TTL**)" are removed. A sample cache configuration is:

proxycache **bdb 10000 1 50 100**

proxycachequeries <queries>

Specify the maximum number of queries to cache. The default is 10000.

proxysavequeries { TRUE | FALSE }

Specify whether the cached queries should be saved across restarts of the caching proxy, to provide hot startup of the cache. Only non-expired queries are reloaded. The default is FALSE.

CAVEAT: of course, the configuration of the proxycache must not change across restarts; the pcache overlay does not perform any consistency checks in this sense. In detail, this option should be disabled unless the existing **proxyattrset** and **proxytemplate** directives are not changed neither in order nor in contents. If new sets and templates are added, or if other details of the pcache overlay configuration changed, this feature should not be affected.

proxyattrset <index> <attrs...>

Used to associate a set of attributes <attrs..> with an <index>. Each attribute set is associated with an integer from 0 to <numattrsets>-1. These indices are used by the **proxytemplate** directive to define cacheable templates. A set of attributes cannot be empty. A set of attributes can contain the special attributes "*" (all user attributes), "+" (all operational attributes) or both; in the latter case, any other attribute is redundant and should be avoided for clarity. A set of attributes can contain "1.1" as the only attribute; in this case, only the presence of the entries is cached.

proxytemplate <template_string> <attrset_index> <ttl> [<negttl>]

Specifies a cacheable template and "time to live" (in sec) <ttl> of queries belonging to the template. An optional <negttl> can be used to specify that negative results (i.e., queries that returned zero entries) should also be cached for the specified number of seconds. Negative results are not cached by default.

response-callback { head | tail }

Specifies whether the response callback should be placed at the **tail** (the default) or at the **head** (actually, wherever the stacking sequence would make it appear) of the callback list. This affects how the overlay interacts with other overlays, since the proxycache overlay should be executed as early as possible (and thus configured as late as possible), to get a chance to return the cached results; however, if executed early at response, it would cache entries that may be later "massaged" by other databases and thus returned *after* massaging the first time, and *before* massaging when cached.

There are some constraints:

all values must be positive;

<entry_limit> must be less than or equal to **<max_entries>**;

<numattrsets> attribute sets SHOULD be defined by using the directive **proxyattrset**;

all attribute sets SHOULD be referenced by (at least) one **proxytemplate** directive;

The following adds a template with filter string **(&(sn=)(givenName=))** and attributes mail, postaladdress, telephonenumber and a TTL of 1 hour.

proxyattrset **0 mail postaladdress telephonenumber**
proxytemplate **(&(sn=)(givenName=)) 0 3600**

Directives for configuring the underlying database must also be given, as shown here:

directory /var/tmp/cache
cachesize 100

Any valid directives for the chosen database type may be used. Indexing should be used as appropriate for the queries being handled. In addition, an equality index on the **queryid** attribute should be configured, to assist in the removal of expired query data.

CAVEATS

Caching data is prone to inconsistencies because updates on the remote server will not be reflected in the response of the cache at least (and at most) for the duration of the **proxytemplate TTL**.

The remote server should expose the **objectClass** attribute because the underlying database that actually caches the entries may need it for optimal local processing of the queries.

Another potential (and subtle) inconsistency may occur when data is retrieved with different identities and specific per-identity access control is enforced by the remote server. If data was retrieved with an identity that collected only partial results because of access rules enforcement on the remote server, other users with different access privileges on the remote server will get different results from the remote server and from the cache. If those users have higher access privileges on the remote server, they will get from the cache only a subset of the results they would get directly from the remote server; but if they have lower access privileges, they will get from the cache a superset of the results they would get directly from the remote server. Either occurrence may or may not be acceptable, based on the security policy of the cache and of the remote server. It is important to note that in this case the proxy is violating the security of the remote server by disclosing to an identity data that was collected by another identity. For this reason, it is suggested that, when using **back-ldap**, proxy caching be used in conjunction with the *identity assertion* feature of **slapd-ldap**(5) (see the **idassert-bind** and the **idassert-authz** statements), so that remote server interrogation occurs with a vanilla identity that has some relatively high **search** and **read** access privileges, and the "real" access control is delegated to the proxy's ACLs. Beware that since only the cached fraction of the real datum is available to the cache, it may not be possible to enforce the same access rules that are defined on the remote server. When security is a concern, cached proxy access must be carefully tailored.

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapd−meta**(5), **slapd−sql**(5), **slapd**(8).

AUTHOR

Originally implemented by Apurva Kumar as an extension to back-meta; turned into an overlay by Howard Chu.

NAME

slapo-pcache − proxycache overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **pcache** overlay to **slapd**(8) allows caching of LDAP search requests (queries) in a local database. For an incoming query, the proxy cache determines its corresponding **template**. If the template was specified as cacheable using the **proxytemplate** directive and the request is contained in a cached request, it is answered from the proxy cache. Otherwise, the search is performed as usual and cacheable search results are saved in the cache for use in future queries.

A template is defined by a filter string and an index identifying a set of attributes. The **template string** for a query can be obtained by removing assertion values from the RFC 4515 representation of its search filter. A query belongs to a template if its template string and set of projected attributes correspond to a cacheable template. Examples of template strings are **(mail=)**, **(|(sn=)(cn=))**, **(&(sn=)(givenName=))**.

The config directives that are specific to the **proxycache** overlay can be prefixed by **proxycache−**, to avoid conflicts with directives specific to the underlying database or to other stacked overlays. This may be particularly useful for those directives that refer to the backend used for local storage. The following cache specific directives can be used to configure the proxy cache:

overlay pcache

This directive adds the proxy cache overlay to the current backend. The proxy cache overlay may be used with any backend but is intended for use with the **ldap**, **meta**, and **sql** backends.

proxycache <database> <max_entries> <numattrsets> <entry_limit> <cc_period>

The directive enables proxy caching in the current backend and sets general cache parameters. A <database> backend will be used internally to maintain the cached entries. The chosen database will need to be configured as well, as shown below. Cache replacement is invoked when the cache size grows to <max_entries> entries and continues till the cache size drops below this size. <numattrsets> should be equal to the number of following **proxyattrset** directives. Queries are cached only if they correspond to a cacheable template (specified by the **proxytemplate** directive) and the number of entries returned is less than <entry_limit>. Consistency check is performed every <cc_period> duration (specified in secs). In each cycle queries with expired "time to live(**TTL**)" are removed. A sample cache configuration is:

proxycache **bdb 10000 1 50 100**

proxycachequeries <queries>

Specify the maximum number of queries to cache. The default is 10000.

proxysavequeries { TRUE | FALSE }

Specify whether the cached queries should be saved across restarts of the caching proxy, to provide hot startup of the cache. Only non-expired queries are reloaded. The default is FALSE.

CAVEAT: of course, the configuration of the proxycache must not change across restarts; the pcache overlay does not perform any consistency checks in this sense. In detail, this option should be disabled unless the existing **proxyattrset** and **proxytemplate** directives are not changed neither in order nor in contents. If new sets and templates are added, or if other details of the pcache overlay configuration changed, this feature should not be affected.

proxyattrset <index> <attrs...>

Used to associate a set of attributes <attrs..> with an <index>. Each attribute set is associated with an integer from 0 to <numattrsets>-1. These indices are used by the **proxytemplate** directive to define cacheable templates. A set of attributes cannot be empty. A set of attributes can contain the special attributes "*" (all user attributes), "+" (all operational attributes) or both; in the latter case, any other attribute is redundant and should be avoided for clarity. A set of attributes can contain "1.1" as the only attribute; in this case, only the presence of the entries is cached.

proxytemplate <template_string> <attrset_index> <ttl> [<negttl>]

Specifies a cacheable template and "time to live" (in sec) <ttl> of queries belonging to the template. An optional <negttl> can be used to specify that negative results (i.e., queries that returned zero entries) should also be cached for the specified number of seconds. Negative results are not cached by default.

response-callback { head | tail }

Specifies whether the response callback should be placed at the **tail** (the default) or at the **head** (actually, wherever the stacking sequence would make it appear) of the callback list. This affects how the overlay interacts with other overlays, since the proxycache overlay should be executed as early as possible (and thus configured as late as possible), to get a chance to return the cached results; however, if executed early at response, it would cache entries that may be later "massaged" by other databases and thus returned *after* massaging the first time, and *before* massaging when cached.

There are some constraints:

all values must be positive;

<entry_limit> must be less than or equal to **<max_entries>**;

<numattrsets> attribute sets SHOULD be defined by using the directive **proxyattrset**;

all attribute sets SHOULD be referenced by (at least) one **proxytemplate** directive;

The following adds a template with filter string **(&(sn=)(givenName=))** and attributes mail, postaladdress, telephonenumber and a TTL of 1 hour.

proxyattrset **0 mail postaladdress telephonenumber**
proxytemplate **(&(sn=)(givenName=)) 0 3600**

Directives for configuring the underlying database must also be given, as shown here:

directory /var/tmp/cache
cachesize 100

Any valid directives for the chosen database type may be used. Indexing should be used as appropriate for the queries being handled. In addition, an equality index on the **queryid** attribute should be configured, to assist in the removal of expired query data.

CAVEATS

Caching data is prone to inconsistencies because updates on the remote server will not be reflected in the response of the cache at least (and at most) for the duration of the **proxytemplate TTL**.

The remote server should expose the **objectClass** attribute because the underlying database that actually caches the entries may need it for optimal local processing of the queries.

Another potential (and subtle) inconsistency may occur when data is retrieved with different identities and specific per-identity access control is enforced by the remote server. If data was retrieved with an identity that collected only partial results because of access rules enforcement on the remote server, other users with different access privileges on the remote server will get different results from the remote server and from the cache. If those users have higher access privileges on the remote server, they will get from the cache only a subset of the results they would get directly from the remote server; but if they have lower access privileges, they will get from the cache a superset of the results they would get directly from the remote server. Either occurrence may or may not be acceptable, based on the security policy of the cache and of the remote server. It is important to note that in this case the proxy is violating the security of the remote server by disclosing to an identity data that was collected by another identity. For this reason, it is suggested that, when using **back-ldap**, proxy caching be used in conjunction with the *identity assertion* feature of **slapd-ldap**(5) (see the **idassert-bind** and the **idassert-authz** statements), so that remote server interrogation occurs with a vanilla identity that has some relatively high **search** and **read** access privileges, and the "real" access control is delegated to the proxy's ACLs. Beware that since only the cached fraction of the real datum is available to the cache, it may not be possible to enforce the same access rules that are defined on the remote server. When security is a concern, cached proxy access must be carefully tailored.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapd−meta**(5), **slapd−sql**(5), **slapd**(8).

AUTHOR

Originally implemented by Apurva Kumar as an extension to back-meta; turned into an overlay by Howard Chu.

NAME

slapo-ppolicy − Password Policy overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **ppolicy** overlay is an implementation of the most recent IETF Password Policy proposal for LDAP. When instantiated, it intercepts, decodes and applies specific password policy controls to overall use of a backend database, changes to user password fields, etc.

The overlay provides a variety of password control mechanisms. They include password aging--both minimum and maximum ages, password reuse and duplication control, account time-outs, mandatory password resets, acceptable password content, and even grace logins. Different groups of users may be associated with different password policies, and there is no limit to the number of password policies that may be created.

Note that some of the policies do not take effect when the operation is performed with the **rootdn** identity; all the operations, when performed with any other identity, may be subjected to constraints, like access control.

Note that the IETF Password Policy proposal for LDAP makes sense when considering a single-valued password attribute, while the userPassword attribute allows multiple values. This implementation enforces a single value for the userPassword attribute, despite its specification.

CONFIGURATION

These **slapd.conf** configuration options apply to the ppolicy overlay. They should appear after the **overlay** directive.

ppolicy_default <policyDN>

Specify the DN of the pwdPolicy object to use when no specific policy is set on a given user's entry. If there is no specific policy for an entry and no default is given, then no policies will be enforced.

ppolicy_hash_cleartext

Specify that cleartext passwords present in Add and Modify requests should be hashed before being stored in the database. This violates the X.500/LDAP information model, but may be needed to compensate for LDAP clients that don't use the Password Modify extended operation to manage passwords. It is recommended that when this option is used that compare, search, and read access be denied to all directory users.

ppolicy_use_lockout

A client will always receive an LDAP **InvalidCredentials** response when Binding to a locked account. By default, when a Password Policy control was provided on the Bind request, a Password Policy response will be included with no special error code set. This option changes the Password Policy response to include the **AccountLocked** error code. Note that sending the **AccountLocked** error code provides useful information to an attacker; sites that are sensitive to security issues should not enable this option.

OBJECT CLASS

The **ppolicy** overlay depends on the **pwdPolicy** object class. The definition of that class is as follows:

```
( 1.3.6.1.4.1.42.2.27.8.2.1
  NAME 'pwdPolicy'
  AUXILIARY
  SUP top
  MUST ( pwdAttribute )
  MAY (
    pwdMinAge $ pwdMaxAge $ pwdInHistory $
```

pwdCheckQuality $ pwdMinLength $
pwdExpireWarning $ pwdGraceAuthnLimit $
pwdLockout $ pwdLockoutDuration $
pwdMaxFailure $ pwdFailureCountInterval $
pwdMustChange $ pwdAllowUserChange $
pwdSafeModify))

This implementation also provides an additional **pwdPolicyChecker** objectclass, used for password quality checking (see below).

(1.3.6.1.4.1.4754.2.99.1
NAME 'pwdPolicyChecker'
AUXILIARY
SUP top
MAY (pwdCheckModule))

Every account that should be subject to password policy control should have a **pwdPolicySubentry** attribute containing the DN of a valid **pwdPolicy** entry, or they can simply use the configured default. In this way different users may be managed according to different policies.

OBJECT CLASS ATTRIBUTES

Each one of the sections below details the meaning and use of a particular attribute of this **pwdPolicy** object class.

pwdAttribute

This attribute contains the name of the attribute to which the password policy is applied. For example, the password policy may be applied to the **userPassword** attribute.

Note: in this implementation, the only value accepted for **pwdAttribute** is *userPassword* .

(1.3.6.1.4.1.42.2.27.8.1.1
NAME 'pwdAttribute'
EQUALITY objectIdentifierMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.38)

pwdMinAge

This attribute contains the number of seconds that must elapse between modifications allowed to the password. If this attribute is not present, zero seconds is assumed (i.e. the password may be modified whenever and however often is desired).

(1.3.6.1.4.1.42.2.27.8.1.2
NAME 'pwdMinAge'
EQUALITY integerMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

pwdMaxAge

This attribute contains the number of seconds after which a modified password will expire. If this attribute is not present, or if its value is zero (0), then passwords will not expire.

(1.3.6.1.4.1.42.2.27.8.1.3
NAME 'pwdMaxAge'
EQUALITY integerMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

pwdInHistory

This attribute is used to specify the maximum number of used passwords that will be stored in the **pwdHistory** attribute. If the **pwdInHistory** attribute is not present, or if its value is zero (0), used passwords will not be stored in **pwdHistory** and thus any previously-used password may be reused. No history checking occurs if the password is being modified by the **rootdn**, although the password is saved in the history.

```
( 1.3.6.1.4.1.42.2.27.8.1.4
  NAME 'pwdInHistory'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdCheckQuality

This attribute indicates if and how password syntax will be checked while a password is being modified or added. If this attribute is not present, or its value is zero (0), no syntax checking will be done. If its value is one (1), the server will check the syntax, and if the server is unable to check the syntax, whether due to a client-side hashed password or some other reason, it will be accepted. If its value is two (2), the server will check the syntax, and if the server is unable to check the syntax it will return an error refusing the password.

```
( 1.3.6.1.4.1.42.2.27.8.1.5
  NAME 'pwdCheckQuality'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdMinLength

When syntax checking is enabled (see also the **pwdCheckQuality** attribute), this attribute contains the minimum number of characters that will be accepted in a password. If this attribute is not present, minimum password length is not enforced. If the server is unable to check the length of the password, whether due to a client-side hashed password or some other reason, the server will, depending on the value of **pwdCheckQuality**, either accept the password without checking it (if **pwdCheckQuality** is zero (0) or one (1)) or refuse it (if **pwdCheckQuality** is two (2)).

```
( 1.3.6.1.4.1.42.2.27.8.1.6
  NAME 'pwdMinLength'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdExpireWarning

This attribute contains the maximum number of seconds before a password is due to expire that expiration warning messages will be returned to a user who is authenticating to the directory. If this attribute is not present, or if the value is zero (0), no warnings will be sent.

```
( 1.3.6.1.4.1.42.2.27.8.1.7
  NAME 'pwdExpireWarning'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdGraceAuthnLimit

This attribute contains the number of times that an expired password may be used to authenticate a user to the directory. If this attribute is not present or if its value is zero (0), users with expired passwords will not be allowed to authenticate to the directory.

```
( 1.3.6.1.4.1.42.2.27.8.1.8
  NAME 'pwdGraceAuthnLimit'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdLockout

This attribute specifies the action that should be taken by the directory when a user has made a number of failed attempts to authenticate to the directory. If **pwdLockout** is set (its value is "TRUE"), the user will not be allowed to attempt to authenticate to the directory after there have been a specified number of consecutive failed bind attempts. The maximum number of consecutive failed bind attempts allowed is specified by the **pwdMaxFailure** attribute. If **pwdLockout** is not present, or if its value is "FALSE", the password may be used to authenticate no matter how many consecutive failed bind attempts have been made.

```
( 1.3.6.1.4.1.42.2.27.8.1.9
  NAME 'pwdLockout'
  EQUALITY booleanMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  SINGLE-VALUE )
```

pwdLockoutDuration

This attribute contains the number of seconds during which the password cannot be used to authenticate the user to the directory due to too many consecutive failed bind attempts. (See also **pwdLockout** and **pwdMaxFailure**.) If **pwdLockoutDuration** is not present, or if its value is zero (0), the password cannot be used to authenticate the user to the directory again until it is reset by an administrator.

```
( 1.3.6.1.4.1.42.2.27.8.1.10
  NAME 'pwdLockoutDuration'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdMaxFailure

This attribute contains the number of consecutive failed bind attempts after which the password may not be used to authenticate a user to the directory. If **pwdMaxFailure** is not present, or its value is zero (0), then a user will be allowed to continue to attempt to authenticate to the directory, no matter how many consecutive failed bind attempts have occurred with that user's DN. (See also **pwdLockout** and **pwdLockoutDuration**.)

```
( 1.3.6.1.4.1.42.2.27.8.1.11
  NAME 'pwdMaxFailure'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdFailureCountInterval

This attribute contains the number of seconds after which old consecutive failed bind attempts are purged from the failure counter, even though no successful authentication has occurred. If **pwdFailureCountInterval** is not present, or its value is zero (0), the failure counter will only be reset by a successful authentication.

```
( 1.3.6.1.4.1.42.2.27.8.1.12
  NAME 'pwdFailureCountInterval'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
```

SINGLE-VALUE)

pwdMustChange

This attribute specifies whether users must change their passwords when they first bind to the directory after a password is set or reset by the administrator, or not. If **pwdMustChange** has a value of "TRUE", users must change their passwords when they first bind to the directory after a password is set or reset by the administrator. If **pwdMustChange** is not present, or its value is "FALSE", users are not required to change their password upon binding after the administrator sets or resets the password.

(1.3.6.1.4.1.42.2.27.8.1.13
NAME 'pwdMustChange'
EQUALITY booleanMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

pwdAllowUserChange

This attribute specifies whether users are allowed to change their own passwords or not. If **pwdAllowUserChange** is set to "TRUE", or if the attribute is not present, users will be allowed to change their own passwords. If its value is "FALSE", users will not be allowed to change their own passwords.

(1.3.6.1.4.1.42.2.27.8.1.14
NAME 'pwdAllowUserChange'
EQUALITY booleanMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

pwdSafeModify

This attribute denotes whether the user's existing password must be sent along with their new password when changing a password. If **pwdSafeModify** is set to "TRUE", the existing password must be sent along with the new password. If the attribute is not present, or its value is "FALSE", the existing password need not be sent along with the new password.

(1.3.6.1.4.1.42.2.27.8.1.15
NAME 'pwdSafeModify'
EQUALITY booleanMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

pwdCheckModule

This attribute names a user-defined loadable module that must instantiate the check_password() function. This function will be called to further check a new password if **pwdCheckQuality** is set to one (1) or two (2), after all of the built-in password compliance checks have been passed. This function will be called according to this function prototype:

int *check_password* (char *pPasswd, char **ppErrStr, Entry *pEntry);

The **pPasswd** parameter contains the clear-text user password, the **ppErrStr** parameter contains a double pointer that allows the function to return human-readable details about any error it encounters. The optional **pEntry** parameter, if non-NULL, carries a pointer to the entry whose password is being checked. If **ppErrStr** is NULL, then *funcName* must NOT attempt to use it/them. A return value of LDAP_SUCCESS from the called function indicates that the password is ok, any other value indicates that the password is unacceptable. If the password is unacceptable, the server will return an error to the client, and **ppErrStr** may be used to return a human-readable textual explanation of the error. The error string must be dynamically allocated as it will be free()'d by slapd.

(1.3.6.1.4.1.4754.1.99.1
NAME 'pwdCheckModule'

EQUALITY caseExactIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
SINGLE-VALUE)

Note: The user-defined loadable module named by **pwdCheckModule** must be in **slapd's** standard executable search PATH.

Note: **pwdCheckModule** is a non-standard extension to the LDAP password policy proposal.

OPERATIONAL ATTRIBUTES

The operational attributes used by the **ppolicy** module are stored in the user's entry. Most of these attributes are not intended to be changed directly by users; they are there to track user activity. They have been detailed here so that administrators and users can both understand the workings of the **ppolicy** module.

Note that the current IETF Password Policy proposal does not define how these operational attributes are expected to behave in a replication environment. In general, authentication attempts on a slave server only affect the copy of the operational attributes on that slave and will not affect any attributes for a user's entry on the master server. Operational attribute changes resulting from authentication attempts on a master server will usually replicate to the slaves (and also overwrite any changes that originated on the slave). These behaviors are not guaranteed and are subject to change when a formal specification emerges.

userPassword

The **userPassword** attribute is not strictly part of the **ppolicy** module. It is, however, the attribute that is tracked and controlled by the module. Please refer to the standard OpenLDAP schema for its definition.

pwdPolicySubentry

This attribute refers directly to the **pwdPolicy** subentry that is to be used for this particular directory user. If **pwdPolicySubentry** exists, it must contain the DN of a valid **pwdPolicy** object. If it does not exist, the **ppolicy** module will enforce the default password policy rules on the user associated with this authenticating DN. If there is no default, or the referenced subentry does not exist, then no policy rules will be enforced.

(1.3.6.1.4.1.42.2.27.8.1.23
NAME 'pwdPolicySubentry'
DESC 'The pwdPolicy subentry in effect for
this object'
EQUALITY distinguishedNameMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
SINGLE-VALUE
NO-USER-MODIFICATION
USAGE directoryOperation)

pwdChangedTime

This attribute denotes the last time that the entry's password was changed. This value is used by the password expiration policy to determine whether the password is too old to be allowed to be used for user authentication. If **pwdChangedTime** does not exist, the user's password will not expire.

(1.3.6.1.4.1.42.2.27.8.1.16
NAME 'pwdChangedTime'
DESC 'The time the password was last changed'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
EQUALITY generalizedTimeMatch
ORDERING generalizedTimeOrderingMatch
SINGLE-VALUE

NO-USER-MODIFICATION
USAGE directoryOperation)

pwdAccountLockedTime

This attribute contains the time that the user's account was locked. If the account has been locked, the password may no longer be used to authenticate the user to the directory. If **pwdAccountLockedTime** is set to 000001010000Z, the user's account has been permanently locked and may only be unlocked by an administrator.

(1.3.6.1.4.1.42.2.27.8.1.17
NAME 'pwdAccountLockedTime'
DESC 'The time an user account was locked'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
EQUALITY generalizedTimeMatch
ORDERING generalizedTimeOrderingMatch
SINGLE-VALUE
NO-USER-MODIFICATION
USAGE directoryOperation)

pwdFailureTime

This attribute contains the timestamps of each of the consecutive authentication failures made upon attempted authentication to this DN (i.e. account). If too many timestamps accumulate here (refer to the **pwdMaxFailure** password policy attribute for details), and the **pwdLockout** password policy attribute is set to "TRUE", the account may be locked. (Please also refer to the **pwdLockout** password policy attribute.) Excess timestamps beyond those allowed by **pwdMaxFailure** may also be purged. If a successful authentication is made to this DN (i.e. to this user account), then **pwdFailureTime** will be cleansed of entries.

(1.3.6.1.4.1.42.2.27.8.1.19
NAME 'pwdFailureTime'
DESC 'The timestamps of the last consecutive
authentication failures'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
EQUALITY generalizedTimeMatch
ORDERING generalizedTimeOrderingMatch
NO-USER-MODIFICATION
USAGE directoryOperation)

pwdHistory

This attribute contains the history of previously used passwords for this DN (i.e. for this user account). The values of this attribute are stored in string format as follows:

pwdHistory=
time "#" syntaxOID "#" length "#" data

time=
GeneralizedTime as specified in section 3.3.13 of [RFC4517]

syntaxOID = numericoid
This is the string representation of the dotted-decimal OID that defines the syntax used to store the password. numericoid is described in section 1.4 of [RFC4512].

length = NumericString

The number of octets in the data. NumericString is described in section 3.3.23 of [RFC4517].

data =
Octets representing the password in the format specified by syntaxOID.

This format allows the server to store and transmit a history of passwords that have been used. In order for equality matching on the values in this attribute to function properly, the time field is in GMT format.

```
( 1.3.6.1.4.1.42.2.27.8.1.20
  NAME 'pwdHistory'
  DESC 'The history of user passwords'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
  EQUALITY octetStringMatch
  NO-USER-MODIFICATION
  USAGE directoryOperation)
```

pwdGraceUseTime This attribute contains the list of timestamps of logins made after the user password in the DN has expired. These post-expiration logins are known as "*grace logins*". If too many *grace logins* have been used (please refer to the **pwdGraceLoginLimit** password policy attribute), then the DN will no longer be allowed to be used to authenticate the user to the directory until the administrator changes the DN's **userPassword** attribute.

```
( 1.3.6.1.4.1.42.2.27.8.1.21
  NAME 'pwdGraceUseTime'
  DESC 'The timestamps of the grace login once the password has expired'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  EQUALITY generalizedTimeMatch
  NO-USER-MODIFICATION
  USAGE directoryOperation)
```

pwdReset

This attribute indicates whether the user's password has been reset by the administrator and thus must be changed upon first use of this DN for authentication to the directory. If **pwdReset** is set to "TRUE", then the password was reset and the user must change it upon first authentication. If the attribute does not exist, or is set to "FALSE", the user need not change their password due to administrative reset.

```
( 1.3.6.1.4.1.42.2.27.8.1.22
  NAME 'pwdReset'
  DESC 'The indication that the password has
    been reset'
  EQUALITY booleanMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  SINGLE-VALUE
  USAGE directoryOperation)
```

EXAMPLES

```
database bdb
suffix dc=example,dc=com
overlay ppolicy
ppolicy_default "cn=Standard,ou=Policies,dc=example,dc=com"
```

SEE ALSO

ldap(3), **slapd.conf**(5),

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

IETF LDAP password policy proposal by P. Behera, L. Poitou and J. Sermersheim: documented in IETF document "draft-behera-ldap-password-policy-09.txt".

BUGS

The LDAP Password Policy specification is not yet an approved standard, and it is still evolving. This code will continue to be in flux until the specification is finalized.

ACKNOWLEDGEMENTS

This module was written in 2004 by Howard Chu of Symas Corporation with significant input from Neil Dunbar and Kartik Subbarao of Hewlett-Packard.

This manual page borrows heavily and shamelessly from the specification upon which the password policy module it describes is based. This source is the IETF LDAP password policy proposal by P. Behera, L. Poitou and J. Sermersheim. The proposal is fully documented in the IETF document named draft-behera-ldap-password-policy-09.txt, written in July of 2005.

NAME

slapo-ppolicy − Password Policy overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **ppolicy** overlay is an implementation of the most recent IETF Password Policy proposal for LDAP. When instantiated, it intercepts, decodes and applies specific password policy controls to overall use of a backend database, changes to user password fields, etc.

The overlay provides a variety of password control mechanisms. They include password aging--both minimum and maximum ages, password reuse and duplication control, account time-outs, mandatory password resets, acceptable password content, and even grace logins. Different groups of users may be associated with different password policies, and there is no limit to the number of password policies that may be created.

Note that some of the policies do not take effect when the operation is performed with the **rootdn** identity; all the operations, when performed with any other identity, may be subjected to constraints, like access control.

Note that the IETF Password Policy proposal for LDAP makes sense when considering a single-valued password attribute, while the userPassword attribute allows multiple values. This implementation enforces a single value for the userPassword attribute, despite its specification.

CONFIGURATION

These **slapd.conf** configuration options apply to the ppolicy overlay. They should appear after the **overlay** directive.

ppolicy_default <policyDN>

Specify the DN of the pwdPolicy object to use when no specific policy is set on a given user's entry. If there is no specific policy for an entry and no default is given, then no policies will be enforced.

ppolicy_hash_cleartext

Specify that cleartext passwords present in Add and Modify requests should be hashed before being stored in the database. This violates the X.500/LDAP information model, but may be needed to compensate for LDAP clients that don't use the Password Modify extended operation to manage passwords. It is recommended that when this option is used that compare, search, and read access be denied to all directory users.

ppolicy_use_lockout

A client will always receive an LDAP **InvalidCredentials** response when Binding to a locked account. By default, when a Password Policy control was provided on the Bind request, a Password Policy response will be included with no special error code set. This option changes the Password Policy response to include the **AccountLocked** error code. Note that sending the **AccountLocked** error code provides useful information to an attacker; sites that are sensitive to security issues should not enable this option.

OBJECT CLASS

The **ppolicy** overlay depends on the **pwdPolicy** object class. The definition of that class is as follows:

```
( 1.3.6.1.4.1.42.2.27.8.2.1
  NAME 'pwdPolicy'
  AUXILIARY
  SUP top
  MUST ( pwdAttribute )
  MAY (
    pwdMinAge $ pwdMaxAge $ pwdInHistory $
```

pwdCheckQuality $ pwdMinLength $
pwdExpireWarning $ pwdGraceAuthnLimit $
pwdLockout $ pwdLockoutDuration $
pwdMaxFailure $ pwdFailureCountInterval $
pwdMustChange $ pwdAllowUserChange $
pwdSafeModify))

This implementation also provides an additional **pwdPolicyChecker** objectclass, used for password quality checking (see below).

(1.3.6.1.4.1.4754.2.99.1
NAME 'pwdPolicyChecker'
AUXILIARY
SUP top
MAY (pwdCheckModule))

Every account that should be subject to password policy control should have a **pwdPolicySubentry** attribute containing the DN of a valid **pwdPolicy** entry, or they can simply use the configured default. In this way different users may be managed according to different policies.

OBJECT CLASS ATTRIBUTES

Each one of the sections below details the meaning and use of a particular attribute of this **pwdPolicy** object class.

pwdAttribute

This attribute contains the name of the attribute to which the password policy is applied. For example, the password policy may be applied to the **userPassword** attribute.

Note: in this implementation, the only value accepted for **pwdAttribute** is *userPassword* .

(1.3.6.1.4.1.42.2.27.8.1.1
NAME 'pwdAttribute'
EQUALITY objectIdentifierMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.38)

pwdMinAge

This attribute contains the number of seconds that must elapse between modifications allowed to the password. If this attribute is not present, zero seconds is assumed (i.e. the password may be modified whenever and however often is desired).

(1.3.6.1.4.1.42.2.27.8.1.2
NAME 'pwdMinAge'
EQUALITY integerMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

pwdMaxAge

This attribute contains the number of seconds after which a modified password will expire. If this attribute is not present, or if its value is zero (0), then passwords will not expire.

(1.3.6.1.4.1.42.2.27.8.1.3
NAME 'pwdMaxAge'
EQUALITY integerMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

pwdInHistory

This attribute is used to specify the maximum number of used passwords that will be stored in the **pwdHistory** attribute. If the **pwdInHistory** attribute is not present, or if its value is zero (0), used passwords will not be stored in **pwdHistory** and thus any previously-used password may be reused. No history checking occurs if the password is being modified by the **rootdn**, although the password is saved in the history.

```
( 1.3.6.1.4.1.42.2.27.8.1.4
  NAME 'pwdInHistory'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdCheckQuality

This attribute indicates if and how password syntax will be checked while a password is being modified or added. If this attribute is not present, or its value is zero (0), no syntax checking will be done. If its value is one (1), the server will check the syntax, and if the server is unable to check the syntax, whether due to a client-side hashed password or some other reason, it will be accepted. If its value is two (2), the server will check the syntax, and if the server is unable to check the syntax it will return an error refusing the password.

```
( 1.3.6.1.4.1.42.2.27.8.1.5
  NAME 'pwdCheckQuality'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdMinLength

When syntax checking is enabled (see also the **pwdCheckQuality** attribute), this attribute contains the minimum number of characters that will be accepted in a password. If this attribute is not present, minimum password length is not enforced. If the server is unable to check the length of the password, whether due to a client-side hashed password or some other reason, the server will, depending on the value of **pwdCheckQuality**, either accept the password without checking it (if **pwdCheckQuality** is zero (0) or one (1)) or refuse it (if **pwdCheckQuality** is two (2)).

```
( 1.3.6.1.4.1.42.2.27.8.1.6
  NAME 'pwdMinLength'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdExpireWarning

This attribute contains the maximum number of seconds before a password is due to expire that expiration warning messages will be returned to a user who is authenticating to the directory. If this attribute is not present, or if the value is zero (0), no warnings will be sent.

```
( 1.3.6.1.4.1.42.2.27.8.1.7
  NAME 'pwdExpireWarning'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdGraceAuthnLimit

This attribute contains the number of times that an expired password may be used to authenticate a user to the directory. If this attribute is not present or if its value is zero (0), users with expired passwords will not be allowed to authenticate to the directory.

```
( 1.3.6.1.4.1.42.2.27.8.1.8
  NAME 'pwdGraceAuthnLimit'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdLockout

This attribute specifies the action that should be taken by the directory when a user has made a number of failed attempts to authenticate to the directory. If **pwdLockout** is set (its value is "TRUE"), the user will not be allowed to attempt to authenticate to the directory after there have been a specified number of consecutive failed bind attempts. The maximum number of consecutive failed bind attempts allowed is specified by the **pwdMaxFailure** attribute. If **pwdLockout** is not present, or if its value is "FALSE", the password may be used to authenticate no matter how many consecutive failed bind attempts have been made.

```
( 1.3.6.1.4.1.42.2.27.8.1.9
  NAME 'pwdLockout'
  EQUALITY booleanMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  SINGLE-VALUE )
```

pwdLockoutDuration

This attribute contains the number of seconds during which the password cannot be used to authenticate the user to the directory due to too many consecutive failed bind attempts. (See also **pwdLockout** and **pwdMaxFailure**.) If **pwdLockoutDuration** is not present, or if its value is zero (0), the password cannot be used to authenticate the user to the directory again until it is reset by an administrator.

```
( 1.3.6.1.4.1.42.2.27.8.1.10
  NAME 'pwdLockoutDuration'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdMaxFailure

This attribute contains the number of consecutive failed bind attempts after which the password may not be used to authenticate a user to the directory. If **pwdMaxFailure** is not present, or its value is zero (0), then a user will be allowed to continue to attempt to authenticate to the directory, no matter how many consecutive failed bind attempts have occurred with that user's DN. (See also **pwdLockout** and **pwdLockoutDuration**.)

```
( 1.3.6.1.4.1.42.2.27.8.1.11
  NAME 'pwdMaxFailure'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

pwdFailureCountInterval

This attribute contains the number of seconds after which old consecutive failed bind attempts are purged from the failure counter, even though no successful authentication has occurred. If **pwdFailureCountInterval** is not present, or its value is zero (0), the failure counter will only be reset by a successful authentication.

```
( 1.3.6.1.4.1.42.2.27.8.1.12
  NAME 'pwdFailureCountInterval'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
```

SINGLE-VALUE)

pwdMustChange

This attribute specifies whether users must change their passwords when they first bind to the directory after a password is set or reset by the administrator, or not. If **pwdMustChange** has a value of "TRUE", users must change their passwords when they first bind to the directory after a password is set or reset by the administrator. If **pwdMustChange** is not present, or its value is "FALSE", users are not required to change their password upon binding after the administrator sets or resets the password.

(1.3.6.1.4.1.42.2.27.8.1.13
NAME 'pwdMustChange'
EQUALITY booleanMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

pwdAllowUserChange

This attribute specifies whether users are allowed to change their own passwords or not. If **pwdAllowUserChange** is set to "TRUE", or if the attribute is not present, users will be allowed to change their own passwords. If its value is "FALSE", users will not be allowed to change their own passwords.

(1.3.6.1.4.1.42.2.27.8.1.14
NAME 'pwdAllowUserChange'
EQUALITY booleanMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

pwdSafeModify

This attribute denotes whether the user's existing password must be sent along with their new password when changing a password. If **pwdSafeModify** is set to "TRUE", the existing password must be sent along with the new password. If the attribute is not present, or its value is "FALSE", the existing password need not be sent along with the new password.

(1.3.6.1.4.1.42.2.27.8.1.15
NAME 'pwdSafeModify'
EQUALITY booleanMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

pwdCheckModule

This attribute names a user-defined loadable module that must instantiate the check_password() function. This function will be called to further check a new password if **pwdCheckQuality** is set to one (1) or two (2), after all of the built-in password compliance checks have been passed. This function will be called according to this function prototype:

int *check_password* (char *pPasswd, char **ppErrStr, Entry *pEntry);

The **pPasswd** parameter contains the clear-text user password, the **ppErrStr** parameter contains a double pointer that allows the function to return human-readable details about any error it encounters. The optional **pEntry** parameter, if non-NULL, carries a pointer to the entry whose password is being checked. If **ppErrStr** is NULL, then *funcName* must NOT attempt to use it/them. A return value of LDAP_SUCCESS from the called function indicates that the password is ok, any other value indicates that the password is unacceptable. If the password is unacceptable, the server will return an error to the client, and **ppErrStr** may be used to return a human-readable textual explanation of the error. The error string must be dynamically allocated as it will be free()'d by slapd.

(1.3.6.1.4.1.4754.1.99.1
NAME 'pwdCheckModule'

```
EQUALITY caseExactIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
SINGLE-VALUE )
```

Note: The user-defined loadable module named by **pwdCheckModule** must be in **slapd's** standard executable search PATH.

Note: **pwdCheckModule** is a non-standard extension to the LDAP password policy proposal.

OPERATIONAL ATTRIBUTES

The operational attributes used by the **ppolicy** module are stored in the user's entry. Most of these attributes are not intended to be changed directly by users; they are there to track user activity. They have been detailed here so that administrators and users can both understand the workings of the **ppolicy** module.

Note that the current IETF Password Policy proposal does not define how these operational attributes are expected to behave in a replication environment. In general, authentication attempts on a slave server only affect the copy of the operational attributes on that slave and will not affect any attributes for a user's entry on the master server. Operational attribute changes resulting from authentication attempts on a master server will usually replicate to the slaves (and also overwrite any changes that originated on the slave). These behaviors are not guaranteed and are subject to change when a formal specification emerges.

userPassword

The **userPassword** attribute is not strictly part of the **ppolicy** module. It is, however, the attribute that is tracked and controlled by the module. Please refer to the standard OpenLDAP schema for its definition.

pwdPolicySubentry

This attribute refers directly to the **pwdPolicy** subentry that is to be used for this particular directory user. If **pwdPolicySubentry** exists, it must contain the DN of a valid **pwdPolicy** object. If it does not exist, the **ppolicy** module will enforce the default password policy rules on the user associated with this authenticating DN. If there is no default, or the referenced subentry does not exist, then no policy rules will be enforced.

```
( 1.3.6.1.4.1.42.2.27.8.1.23
  NAME 'pwdPolicySubentry'
  DESC 'The pwdPolicy subentry in effect for
    this object'
  EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  SINGLE-VALUE
  NO-USER-MODIFICATION
  USAGE directoryOperation)
```

pwdChangedTime

This attribute denotes the last time that the entry's password was changed. This value is used by the password expiration policy to determine whether the password is too old to be allowed to be used for user authentication. If **pwdChangedTime** does not exist, the user's password will not expire.

```
( 1.3.6.1.4.1.42.2.27.8.1.16
  NAME 'pwdChangedTime'
  DESC 'The time the password was last changed'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  SINGLE-VALUE
```

```
    NO-USER-MODIFICATION
    USAGE directoryOperation)
```

pwdAccountLockedTime

This attribute contains the time that the user's account was locked. If the account has been locked, the password may no longer be used to authenticate the user to the directory. If **pwdAccountLockedTime** is set to 000001010000Z, the user's account has been permanently locked and may only be unlocked by an administrator.

```
( 1.3.6.1.4.1.42.2.27.8.1.17
  NAME 'pwdAccountLockedTime'
  DESC 'The time an user account was locked'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  SINGLE-VALUE
  NO-USER-MODIFICATION
  USAGE directoryOperation)
```

pwdFailureTime

This attribute contains the timestamps of each of the consecutive authentication failures made upon attempted authentication to this DN (i.e. account). If too many timestamps accumulate here (refer to the **pwdMaxFailure** password policy attribute for details), and the **pwdLockout** password policy attribute is set to "TRUE", the account may be locked. (Please also refer to the **pwdLockout** password policy attribute.) Excess timestamps beyond those allowed by **pwdMaxFailure** may also be purged. If a successful authentication is made to this DN (i.e. to this user account), then **pwdFailureTime** will be cleansed of entries.

```
( 1.3.6.1.4.1.42.2.27.8.1.19
  NAME 'pwdFailureTime'
  DESC 'The timestamps of the last consecutive
    authentication failures'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  NO-USER-MODIFICATION
  USAGE directoryOperation )
```

pwdHistory

This attribute contains the history of previously used passwords for this DN (i.e. for this user account). The values of this attribute are stored in string format as follows:

pwdHistory=
    time "#" syntaxOID "#" length "#" data

time=
    GeneralizedTime as specified in section 3.3.13 of [RFC4517]

syntaxOID = numericoid
    This is the string representation of the dotted-decimal OID that defines the syntax used to store the password. numericoid is described in section 1.4 of [RFC4512].

length = NumericString

The number of octets in the data. NumericString is described in section 3.3.23 of [RFC4517].

data =
Octets representing the password in the format specified by syntaxOID.

This format allows the server to store and transmit a history of passwords that have been used. In order for equality matching on the values in this attribute to function properly, the time field is in GMT format.

```
( 1.3.6.1.4.1.42.2.27.8.1.20
  NAME 'pwdHistory'
  DESC 'The history of user passwords'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
  EQUALITY octetStringMatch
  NO-USER-MODIFICATION
  USAGE directoryOperation)
```

pwdGraceUseTime This attribute contains the list of timestamps of logins made after the user password in the DN has expired. These post-expiration logins are known as "*grace logins*". If too many *grace logins* have been used (please refer to the **pwdGraceLoginLimit** password policy attribute), then the DN will no longer be allowed to be used to authenticate the user to the directory until the administrator changes the DN's **userPassword** attribute.

```
( 1.3.6.1.4.1.42.2.27.8.1.21
  NAME 'pwdGraceUseTime'
  DESC 'The timestamps of the grace login once the password has expired'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  EQUALITY generalizedTimeMatch
  NO-USER-MODIFICATION
  USAGE directoryOperation)
```

pwdReset

This attribute indicates whether the user's password has been reset by the administrator and thus must be changed upon first use of this DN for authentication to the directory. If **pwdReset** is set to "TRUE", then the password was reset and the user must change it upon first authentication. If the attribute does not exist, or is set to "FALSE", the user need not change their password due to administrative reset.

```
( 1.3.6.1.4.1.42.2.27.8.1.22
  NAME 'pwdReset'
  DESC 'The indication that the password has
    been reset'
  EQUALITY booleanMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  SINGLE-VALUE
  USAGE directoryOperation)
```

EXAMPLES

```
database bdb
suffix dc=example,dc=com
overlay ppolicy
ppolicy_default "cn=Standard,ou=Policies,dc=example,dc=com"
```

SEE ALSO

ldap(3), **slapd.conf**(5),

"OpenLDAP Administrator's Guide" (http://www.OpenLDAP.org/doc/admin/)

IETF LDAP password policy proposal by P. Behera, L. Poitou and J. Sermersheim: documented in IETF document "draft-behera-ldap-password-policy-09.txt".

BUGS

The LDAP Password Policy specification is not yet an approved standard, and it is still evolving. This code will continue to be in flux until the specification is finalized.

ACKNOWLEDGEMENTS

This module was written in 2004 by Howard Chu of Symas Corporation with significant input from Neil Dunbar and Kartik Subbarao of Hewlett-Packard.

This manual page borrows heavily and shamelessly from the specification upon which the password policy module it describes is based. This source is the IETF LDAP password policy proposal by P. Behera, L. Poitou and J. Sermersheim. The proposal is fully documented in the IETF document named draft-behera-ldap-password-policy-09.txt, written in July of 2005.

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapo-refint − Referential Integrity overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Referential Integrity overlay can be used with a backend database such as **slapd-bdb**(5) to maintain the cohesiveness of a schema which utilizes reference attributes.

Integrity is maintained by updating database records which contain the named attributes to match the results of a **modrdn** or **delete** operation. For example, if the integrity attribute were configured as **manager**, deletion of the record "uid=robert,ou=people,dc=example,dc=com" would trigger a search for all other records which have a **manager** attribute containing that DN. Entries matching that search would have their **manager** attribute removed. Or, renaming the same record into "uid=george,ou=people,dc=example,dc=com" would trigger a search for all other records which have a **manager** attribute containing that DN. Entries matching that search would have their **manager** attribute deleted and replaced by the new DN.

CONFIGURATION

These **slapd.conf** options apply to the Referential Integrity overlay. They should appear after the **overlay** directive.

refint_attributes <attribute...>

Specify one or more attributes for which integrity will be maintained as described above.

refint_nothing <string>

Specify an arbitrary value to be used as a placeholder when the last value would otherwise be deleted from an attribute. This can be useful in cases where the schema requires the existence of an attribute for which referential integrity is enforced. The attempted deletion of a required attribute will otherwise result in an Object Class Violation, causing the request to fail.

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

NAME

slapo-refint − Referential Integrity overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Referential Integrity overlay can be used with a backend database such as **slapd-bdb**(5) to maintain the cohesiveness of a schema which utilizes reference attributes.

Integrity is maintained by updating database records which contain the named attributes to match the results of a **modrdn** or **delete** operation. For example, if the integrity attribute were configured as **manager**, deletion of the record "uid=robert,ou=people,dc=example,dc=com" would trigger a search for all other records which have a **manager** attribute containing that DN. Entries matching that search would have their **manager** attribute removed. Or, renaming the same record into "uid=george,ou=people,dc=example,dc=com" would trigger a search for all other records which have a **manager** attribute containing that DN. Entries matching that search would have their **manager** attribute deleted and replaced by the new DN.

CONFIGURATION

These **slapd.conf** options apply to the Referential Integrity overlay. They should appear after the **overlay** directive.

refint_attributes <attribute...>

Specify one or more attributes for which integrity will be maintained as described above.

refint_nothing <string>

Specify an arbitrary value to be used as a placeholder when the last value would otherwise be deleted from an attribute. This can be useful in cases where the schema requires the existence of an attribute for which referential integrity is enforced. The attempted deletion of a required attribute will otherwise result in an Object Class Violation, causing the request to fail.

FILES

/etc/openldap/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapo-retcode − return code overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **retcode** overlay to **slapd**(8) is useful to test the behavior of clients when server-generated erroneous and/or unusual responses occur, e.g. error codes, referrals, excessive response times and so on.

The error responses are generated according to different strategies.

In the first case, all operations targeted at a specific configurable subtree cause the object related to the request DN to be looked up and checked for return code data: a response code, plus an optional textual message, an optional configurable delay, an optional matched DN field, and, when the response code is "referral", a (list of) referral(s).

Well-known response codes from standard track documents are provided in **retcode.conf**, which can be included after instantiating the overlay.

In the second case, objects of classes inherited from the **errAbsObject**, like **errObject** or **errAuxObject**, when returned as intermediate responses of a search request, are changed into the response dictated by their content.

A third mode causes objects to be looked up from the underlying database to discover if their class inherits from **errABsObject**; in that case, their content is used to compute the corresponding response.

The behavior is disabled by using the **manageDSAit** control (RFC 3296); in that case, the resulting object, either present in the directory or dynamically generated by the overlay, or contained in the request, is handled as usual.

The config directives that are specific to the **retcode** overlay must be prefixed by **retcode−**, to avoid conflicts with directives specific to the underlying database or to other stacked overlays. The following specific directives can be used to configure the retcode overlay:

retcode−parent <DN>
This directive defines the parent DN where dynamically generated entries reside. If not defined, the suffix of the database is used.

retcode−item <RDN> <errCode> [op=<oplist>] [text=<message>] [ref=<referral>] [sleeptime=<sec>] [matched=<DN>] [unsolicited=<OID>[:<data>]] [flags=[{pre|post}-]disconnect[,...]]
A dynamically generated entry, located below **retcode−parent**. The **errCode** is the number of the response code; it can be in any format supported by **strtol**(3). The optional **oplist** is a list of operations that cause response code generation; if absent, all operations are affected. The **matched** field is the matched DN that is returned along with the error, while the **text** field is an optional diagnostics message. The **ref** field is only allowed for the **referral** response code. The **sleeptime** field causes **slapd**(8) to sleep the specified number of seconds before proceeding with the operation. The **unsolicited** field can be used to cause the return of an RFC 4511 unsolicited response message; if **OID** is not "0", an extended response is generated, with the optional **data** appended. If **flags** contains **disconnect**, or **pre-disconnect**, **slapd**(8) disconnects abruptly, without notice; **post-disconnect** causes disconnection right after sending response as appropriate.

retcode−indir
Enables exploitation of in-directory stored errAbsObject. May result in a lot of unnecessary overhead.

retcode−sleep [-]<n>
Defines a sleep time in seconds that is spent before actually handling any operation. If negative, a random time between 0 and the absolute value of the argument is used.

SCHEMA

The **retcode** overlay utilizes the "return code" schema described herein. This schema is specifically designed for use with this overlay and is not intended to be used otherwise. It is also noted that the schema describe here is *a work in progress*, and hence subject to change without notice. The schema is loaded automatically by the overlay.

The schema includes a number of object classes and associated attribute types as described below.

The error code:

```
( 1.3.6.1.4.1.4203.666.11.4.1.1
  NAME ( 'errCode' )
  DESC 'LDAP error code'
  EQUALITY integerMatch
  ORDERING integerOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

The operations that trigger the response code:

```
( 1.3.6.1.4.1.4203.666.11.4.1.2
  NAME ( 'errOp' )
  DESC 'Operations the errObject applies to'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

The text message:

```
( 1.3.6.1.4.1.4203.666.11.4.1.3
  NAME ( 'errText' )
  DESC 'LDAP error textual description'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )
```

The sleep time before the response is actually returned to the client:

```
( 1.3.6.1.4.1.4203.666.11.4.1.4
  NAME ( 'errSleepTime' )
  DESC 'Time to wait before returning the error'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

The matched DN returned to the client:

```
( 1.3.6.1.4.1.4203.666.11.4.1.5
  NAME ( 'errMatchedDN' )
  DESC 'Value to be returned as matched DN'
  EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  SINGLE-VALUE )
```

The OID to be returned as extended response OID in RFC 4511 unsolicited responses ("0" generates a regular response with msgid set to 0):

```
( 1.3.6.1.4.1.4203.666.11.4.1.6
  NAME ( 'errUnsolicitedOID' )
  DESC 'OID to be returned within unsolicited response'
  EQUALITY objectIdentifierMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
```

SINGLE-VALUE)

The octet string to be returned as extended response data in RFC 4511 unsolicited response:

(1.3.6.1.4.1.4203.666.11.4.1.7
NAME ('errUnsolicitedData')
DESC 'Data to be returned within unsolicited response'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
SINGLE-VALUE)

If TRUE, **slapd**(8) disconnects abruptly without notice; if FALSE, it disconnects after sending response as appropriate:

(1.3.6.1.4.1.4203.666.11.4.1.8
NAME ('errDisconnect')
DESC 'Disconnect without notice'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

The abstract class that triggers the overlay:

(1.3.6.1.4.1.4203.666.11.4.3.0
NAME ('errAbsObject')
SUP top ABSTRACT
MUST (errCode)
MAY (cn $ description $ errOp $ errText $ errSleepTime
$ errMatchedDN))

The standalone structural objectclass for specifically created data:

(1.3.6.1.4.1.4203.666.11.4.3.1
NAME ('errObject')
SUP errAbsObject STRUCTURAL)

The auxiliary objectclass to alter the behavior of existing objects:

(1.3.6.1.4.1.4203.666.11.4.3.2
NAME ('errAuxObject')
SUP errAbsObject AUXILIARY)

EXAMPLE

```
overlay        retcode
retcode-parent  "ou=RetCodes,dc=example,dc=com"
include        ./retcode.conf

# Wait 10 seconds, then return success (0x00)
retcode-item    "cn=Success after 10 seconds" 0x00 sleeptime=10
# Wait 10 seconds, then return timelimitExceeded (0x03)
retcode-item    "cn=Timelimit after 10 seconds" 0x03 sleeptime=10
```

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8),

ACKNOWLEDGEMENTS

This module was written in 2005 by Pierangelo Masarati for SysNet s.n.c.

NAME

slapo-retcode − return code overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **retcode** overlay to **slapd**(8) is useful to test the behavior of clients when server-generated erroneous and/or unusual responses occur, e.g. error codes, referrals, excessive response times and so on.

The error responses are generated according to different strategies.

In the first case, all operations targeted at a specific configurable subtree cause the object related to the request DN to be looked up and checked for return code data: a response code, plus an optional textual message, an optional configurable delay, an optional matched DN field, and, when the response code is "referral", a (list of) referral(s).

Well-known response codes from standard track documents are provided in **retcode.conf**, which can be included after instantiating the overlay.

In the second case, objects of classes inherited from the **errAbsObject**, like **errObject** or **errAuxObject**, when returned as intermediate responses of a search request, are changed into the response dictated by their content.

A third mode causes objects to be looked up from the underlying database to discover if their class inherits from **errABsObject**; in that case, their content is used to compute the corresponding response.

The behavior is disabled by using the **manageDSAit** control (RFC 3296); in that case, the resulting object, either present in the directory or dynamically generated by the overlay, or contained in the request, is handled as usual.

The config directives that are specific to the **retcode** overlay must be prefixed by **retcode−**, to avoid conflicts with directives specific to the underlying database or to other stacked overlays. The following specific directives can be used to configure the retcode overlay:

retcode−parent <DN>
This directive defines the parent DN where dynamically generated entries reside. If not defined, the suffix of the database is used.

retcode−item <RDN> <errCode> [op=<oplist>] [text=<message>] [ref=<referral>] [sleeptime=<sec>] [matched=<DN>] [unsolicited=<OID>[:<data>]] [flags=[{pre|post}-]disconnect[,...]]
A dynamically generated entry, located below **retcode−parent**. The **errCode** is the number of the response code; it can be in any format supported by **strtol**(3). The optional **oplist** is a list of operations that cause response code generation; if absent, all operations are affected. The **matched** field is the matched DN that is returned along with the error, while the **text** field is an optional diagnostics message. The **ref** field is only allowed for the **referral** response code. The **sleeptime** field causes **slapd**(8) to sleep the specified number of seconds before proceeding with the operation. The **unsolicited** field can be used to cause the return of an RFC 4511 unsolicited response message; if **OID** is not "0", an extended response is generated, with the optional **data** appended. If **flags** contains **disconnect**, or **pre-disconnect**, **slapd**(8) disconnects abruptly, without notice; **post-disconnect** causes disconnection right after sending response as appropriate.

retcode−indir
Enables exploitation of in-directory stored errAbsObject. May result in a lot of unnecessary overhead.

retcode−sleep [-]<n>
Defines a sleep time in seconds that is spent before actually handling any operation. If negative, a random time between 0 and the absolute value of the argument is used.

SCHEMA

The **retcode** overlay utilizes the "return code" schema described herein. This schema is specifically designed for use with this overlay and is not intended to be used otherwise. It is also noted that the schema describe here is *a work in progress*, and hence subject to change without notice. The schema is loaded automatically by the overlay.

The schema includes a number of object classes and associated attribute types as described below.

The error code:

```
( 1.3.6.1.4.1.4203.666.11.4.1.1
  NAME ( 'errCode' )
  DESC 'LDAP error code'
  EQUALITY integerMatch
  ORDERING integerOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

The operations that trigger the response code:

```
( 1.3.6.1.4.1.4203.666.11.4.1.2
  NAME ( 'errOp' )
  DESC 'Operations the errObject applies to'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

The text message:

```
( 1.3.6.1.4.1.4203.666.11.4.1.3
  NAME ( 'errText' )
  DESC 'LDAP error textual description'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )
```

The sleep time before the response is actually returned to the client:

```
( 1.3.6.1.4.1.4203.666.11.4.1.4
  NAME ( 'errSleepTime' )
  DESC 'Time to wait before returning the error'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )
```

The matched DN returned to the client:

```
( 1.3.6.1.4.1.4203.666.11.4.1.5
  NAME ( 'errMatchedDN' )
  DESC 'Value to be returned as matched DN'
  EQUALITY distinguishedNameMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  SINGLE-VALUE )
```

The OID to be returned as extended response OID in RFC 4511 unsolicited responses ("0" generates a regular response with msgid set to 0):

```
( 1.3.6.1.4.1.4203.666.11.4.1.6
  NAME ( 'errUnsolicitedOID' )
  DESC 'OID to be returned within unsolicited response'
  EQUALITY objectIdentifierMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
```

SINGLE-VALUE)

The octet string to be returned as extended response data in RFC 4511 unsolicited response:

(1.3.6.1.4.1.4203.666.11.4.1.7
NAME ('errUnsolicitedData')
DESC 'Data to be returned within unsolicited response'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
SINGLE-VALUE)

If TRUE, **slapd**(8) disconnects abruptly without notice; if FALSE, it disconnects after sending response as appropriate:

(1.3.6.1.4.1.4203.666.11.4.1.8
NAME ('errDisconnect')
DESC 'Disconnect without notice'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
SINGLE-VALUE)

The abstract class that triggers the overlay:

(1.3.6.1.4.1.4203.666.11.4.3.0
NAME ('errAbsObject')
SUP top ABSTRACT
MUST (errCode)
MAY (cn $ description $ errOp $ errText $ errSleepTime
$ errMatchedDN))

The standalone structural objectclass for specifically created data:

(1.3.6.1.4.1.4203.666.11.4.3.1
NAME ('errObject')
SUP errAbsObject STRUCTURAL)

The auxiliary objectclass to alter the behavior of existing objects:

(1.3.6.1.4.1.4203.666.11.4.3.2
NAME ('errAuxObject')
SUP errAbsObject AUXILIARY)

EXAMPLE

```
overlay        retcode
retcode-parent  "ou=RetCodes,dc=example,dc=com"
include        ./retcode.conf

# Wait 10 seconds, then return success (0x00)
retcode-item    "cn=Success after 10 seconds" 0x00 sleeptime=10
# Wait 10 seconds, then return timelimitExceeded (0x03)
retcode-item    "cn=Timelimit after 10 seconds" 0x03 sleeptime=10
```

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd**(8),

ACKNOWLEDGEMENTS

This module was written in 2005 by Pierangelo Masarati for SysNet s.n.c.

NAME

slapo-rwm − rewrite/remap overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The **rwm** overlay to **slapd**(8) performs basic DN/data rewrite and objectClass/attributeType mapping. Its usage is mostly intended to provide virtual views of existing data either remotely, in conjunction with the proxy backend described in **slapd-ldap**(5), or locally, in conjunction with the relay backend described in **slapd-relay**(5).

This overlay is experimental.

MAPPING

An important feature of the **rwm** overlay is the capability to map objectClasses and attributeTypes from the local set (or a subset of it) to a foreign set, and vice versa. This is accomplished by means of the **rwm-map** directive.

rwm-map {attribute | objectclass} [<local name> | *] {<foreign name> | *}

Map attributeTypes and objectClasses from the foreign server to different values on the local slapd. The reason is that some attributes might not be part of the local slapd's schema, some attribute names might be different but serve the same purpose, etc. If local or foreign name is '*', the name is preserved. If local name is omitted, the foreign name is removed. Unmapped names are preserved if both local and foreign name are '*', and removed if local name is omitted and foreign name is '*'.

The local *objectClasses* and *attributeTypes* must be defined in the local schema; the foreign ones do not have to, but users are encouraged to explicitly define the remote attributeTypes and the objectClasses they intend to map. All in all, when remapping a remote server via back-ldap (**slapd-ldap**(5)) or back-meta (**slapd-meta**(5)) their definition can be easily obtained by querying the *subschemaSubentry* of the remote server; the problem should not exist when remapping a local database. Note, however, that the decision whether to rewrite or not attributeTypes with *distinguishedName syntax*, requires the knowledge of the attributeType syntax. See the REWRITING section for details.

Note that when mapping DN-valued attributes from local to remote, first the DN is rewritten, and then the attributeType is mapped; while mapping from remote to local, first the attributeType is mapped, and then the DN is rewritten. As such, it is important that the local attributeType is appropriately defined as using the distinguishedName syntax. Also, note that there are DN-related syntaxes (i.e. compound types with a portion that is DN-valued), like nameAndOptionalUID, whose values are currently not rewritten.

If the foreign type of an attribute mapping is not defined on the local server, it might be desirable to have the attribute values normalized after the mapping process. Not normalizing the values can lead to wrong results, when the **rwm** overlay is used together with e.g. the **pcache** overlay. This normalization can be enabled by means of the **rwm-normalize-mapped-attrs** directive.

rwm-normalize-mapped-attrs {yes|no}

Set this to "yes", if the **rwm** overlay should try to normalize the values of attributes that are mapped from an attribute type that is unknown to the local server. The default value of this setting is "no".

SUFFIX MASSAGING

A basic feature of the **rwm** overlay is the capability to perform suffix massaging between a virtual and a real naming context by means of the **rwm-suffixmassage** directive. This, in conjunction with proxy backends, **slapd-ldap**(5) and **slapd-meta**(5), or with the relay backend, **slapd-relay**(5), allows to create virtual views of databases. A distinguishing feature of this overlay is that, when instantiated before any database, it can modify the DN of requests *before* database selection. For this reason, rules that rewrite the empty DN ("") or the subschemaSubentry DN (usually "cn=subschema"), would prevent clients from reading the root DSE or the DSA's schema.

rwm-suffixmassage [<virtual naming context>] <real naming context>

Shortcut to implement naming context rewriting; the trailing part of the DN is rewritten from the virtual to the real naming context in the bindDN, searchDN, searchFilterAttrDN, compareDN, compareAttrDN, addDN, addAttrDN, modifyDN, modifyAttrDN, modrDN, newSuperiorDN, deleteDN, exopPasswdDN, and from the real to the virtual naming context in the searchEntryDN, searchAttrDN and matchedDN rewrite contexts. By default no rewriting occurs for the searchFilter and for the referralAttrDN and referralDN rewrite contexts. If no *<virtual naming context>* is given, the first suffix of the database is used; this requires the **rwm-suffixmassage** directive be defined *after* the database **suffix** directive. The **rwm-suffixmassage** directive automatically sets the **rwm-rewriteEngine** to **ON**.

See the REWRITING section for details.

REWRITING

A string is rewritten according to a set of rules, called a 'rewrite context'. The rules are based on POSIX (''extended'') regular expressions with substring matching; basic variable substitution and map resolution of substrings is allowed by specific mechanisms detailed in the following. The behavior of pattern matching/substitution can be altered by a set of flags.

<rewrite context> ::= <rewrite rule> [...]
<rewrite rule> ::= <pattern> <action> [<flags>]

The underlying concept is to build a lightweight rewrite module for the slapd server (initially dedicated to the LDAP backend):

Passes

An incoming string is matched against a set of *rewriteRules*. Rules are made of a *regex match pattern*, a *substitution pattern* and a set of actions, described by a set of *optional flags*. In case of match, string rewriting is performed according to the substitution pattern that allows to refer to substrings matched in the incoming string. The actions, if any, are finally performed. Each rule is executed recursively, unless altered by specific action flags; see "Action Flags" for details. A default limit on the recursion level is set, and can be altered by the **rwm-rewriteMaxPasses** directive, as detailed in the "Additional Configuration Syntax" section. The substitution pattern allows map resolution of substrings. A map is a generic object that maps a substitution pattern to a value. The flags are divided in "Pattern Matching Flags" and "Action Flags"; the former alter the regex match pattern behavior, while the latter alter the actions that are taken after substitution.

Pattern Matching Flags

'C' honors case in matching (default is case insensitive)

'R' use POSIX ''basic'' regular expressions (default is ''extended'')

'M{n}' allow no more than **n** recursive passes for a specific rule; does not alter the max total count of passes, so it can only enforce a stricter limit for a specific rule.

Action Flags

':' apply the rule once only (default is recursive)

'@' stop applying rules in case of match; the current rule is still applied recursively; combine with ':' to apply the current rule only once and then stop.

'#' stop current operation if the rule matches, and issue an 'unwilling to perform' error.

'G{n}' jump **n** rules back and forth (watch for loops!). Note that 'G{1}' is implicit in every rule.

'I' ignores errors in rule; this means, in case of error, e.g. issued by a map, the error is treated as a missed match. The 'unwilling to perform' is not overridden.

'U{n}' uses **n** as return code if the rule matches; the flag does not alter the recursive behavior of the rule, so, to have it performed only once, it must be used in combination with ':', e.g. **':U{32}'** returns the value '32' (indicating noSuchObject) after exactly one execution of the rule, if the pattern matches. As a consequence, its behavior is equivalent to '@', with the return code set to **n**; or, in

other words, '@' is equivalent to 'U{0}'. Positive errors are allowed, indicating the related LDAP error codes as specified in *draft-ietf-ldapbis-protocol*.

The ordering of the flags can be significant. For instance: 'IG{2}' means ignore errors and jump two lines ahead both in case of match and in case of error, while 'G{2}I' means ignore errors, but jump two lines ahead only in case of match.

More flags (mainly Action Flags) will be added as needed.

Pattern Matching

See **regex**(7) and/or **re_format**(7).

Substitution Pattern Syntax

Everything starting with '$' requires substitution;

the only obvious exception is '$$', which is turned into a single '$';

the basic substitution is '$<d>', where '<d>' is a digit; 0 means the whole string, while 1-9 is a submatch, as discussed in **regex**(7) and/or **re_format**(7).

a '$' followed by a '{' invokes an advanced substitution. The pattern is:

'$' '{' [<operator>] <name> '(' <substitution> ')' '}'

where <name> must be a legal name for the map, i.e.

<name> ::= [a-z][a-z0-9]* (case insensitive)
<operator> ::= '>' '|' '&' '&&' '*' '**' '$'

and <substitution> must be a legal substitution pattern, with no limits on the nesting level.

The operators are:

> sub-context invocation; <name> must be a legal, already defined rewrite context name

| external command invocation; <name> must refer to a legal, already defined command name (NOT IMPLEMENTED YET)

& variable assignment; <name> defines a variable in the running operation structure which can be dereferenced later; operator **&** assigns a variable in the rewrite context scope; operator **&&** assigns a variable that scopes the entire session, e.g. its value can be dereferenced later by other rewrite contexts

***** variable dereferencing; <name> must refer to a variable that is defined and assigned for the running operation; operator ***** dereferences a variable scoping the rewrite context; operator ****** dereferences a variable scoping the whole session, e.g. the value is passed across rewrite contexts

$ parameter dereferencing; <name> must refer to an existing parameter; the idea is to make some run-time parameters set by the system available to the rewrite engine, as the client host name, the bind DN if any, constant parameters initialized at config time, and so on; no parameter is currently set by either **back−ldap** or **back−meta**, but constant parameters can be defined in the configuration file by using the **rewriteParam** directive.

Substitution escaping has been delegated to the '$' symbol, which is used instead of '\' in string substitution patterns because '\' is already escaped by slapd's low level parsing routines; as a consequence, regex escaping requires two '\' symbols, e.g. '**.*\.foo\.bar**' must be written as '**.*\\.foo\\.bar**'.

Rewrite Context

A rewrite context is a set of rules which are applied in sequence. The basic idea is to have an application initialize a rewrite engine (think of Apache's mod_rewrite ...) with a set of rewrite contexts; when string rewriting is required, one invokes the appropriate rewrite context with the input string and obtains the newly rewritten one if no errors occur.

Each basic server operation is associated to a rewrite context; they are divided in two main groups: client −> server and server −> client rewriting.

client -> server:

(default) if defined and no specific context is available
bindDN bind
searchDN search
searchFilter search
searchFilterAttrDN search
compareDN compare
compareAttrDN compare AVA
addDN add
addAttrDN add AVA (DN portion of "ref" excluded)
modifyDN modify
modifyAttrDN modify AVA (DN portion of "ref" excluded)
referralAttrDN add/modify DN portion of referrals (default to none)
modrDN modrdn
newSuperiorDN modrdn
deleteDN delete
exopPasswdDN password modify extended operation DN

server -> client:

searchEntryDN search (only if defined; no default; acts on DN of search entries)
searchAttrDN search AVA (only if defined; defaults to searchEntryDN; acts on DN-syntax attributes of search results)
matchedDN all ops (only if applicable; defaults to searchEntryDN)
referralDN all ops (only if applicable; defaults to none)

Basic Configuration Syntax

All rewrite/remap directives start with the prefix **rwm-**; for backwards compatibility with the historical **slapd-ldap**(5) and **slapd-meta**(5) builtin rewrite/remap capabilities, the prefix may be omitted, but this practice is strongly discouraged.

rwm-rewriteEngine { on | off }
If 'on', the requested rewriting is performed; if 'off', no rewriting takes place (an easy way to stop rewriting without altering too much the configuration file).

rwm-rewriteContext <context name> [alias <aliased context name>]
<Context name> is the name that identifies the context, i.e. the name used by the application to refer to the set of rules it contains. It is used also to reference sub contexts in string rewriting. A context may alias another one. In this case the alias context contains no rule, and any reference to it will result in accessing the aliased one.

rwm-rewriteRule <regex match pattern> <substitution pattern> [<flags>]
Determines how a string can be rewritten if a pattern is matched. Examples are reported below.

Additional Configuration Syntax

rwm-rewriteMap <map type> <map name> [<map attrs>]
Allows to define a map that transforms substring rewriting into something else. The map is referenced inside the substitution pattern of a rule.

rwm-rewriteParam <param name> <param value>
Sets a value with global scope, that can be dereferenced by the command '${$paramName}'.

rwm-rewriteMaxPasses <number of passes> [<number of passes per rule>]
Sets the maximum number of total rewriting passes that can be performed in a single rewrite operation (to avoid loops). A safe default is set to 100; note that reaching this limit is still treated as a success; recursive invocation of rules is simply interrupted. The count applies to the rewriting operation as a whole, not to any single rule; an optional per-rule limit can be set. This limit is overridden by setting specific per-rule limits with the 'M{n}' flag.

MAPS

Currently, few maps are builtin and there are no provisions for developers to register new map types at runtime.

Supported maps are:

LDAP <URI> [bindwhen=<when>] [version=<version>] [binddn=<DN>] [credentials=<cred>]
The **LDAP** map expands a value by performing a simple LDAP search. Its configuration is based on a mandatory URI, whose **attrs** portion must contain exactly one attribute (use **entryDN** to fetch the DN of an entry). If a multi-valued attribute is used, only the first value is considered.

The parameter **bindwhen** determines when the connection is established. It can take the values **now**, **later**, and **everytime**, respectively indicating that the connection should be created at startup, when required, or any time it is used. In the former two cases, the connection is cached, while in the latter a fresh new one is used all times. This is the default.

The parameters **binddn** and **credentials** represent the DN and the password that is used to perform an authenticated simple bind before performing the search operation; if not given, an anonymous connection is used.

The parameter **version** can be 2 or 3 to indicate the protocol version that must be used. The default is 3.

REWRITE CONFIGURATION EXAMPLES

```
# set to 'off' to disable rewriting
rwm-rewriteEngine on

# the rules the "suffixmassage" directive implies
rwm-rewriteEngine on
# all dataflow from client to server referring to DNs
rwm-rewriteContext default
rwm-rewriteRule "(.+,)?<virtualnamingcontext>$" "$1<realnamingcontext>" ":"
# empty filter rule
rwm-rewriteContext searchFilter
# all dataflow from server to client
rwm-rewriteContext searchEntryDN
rwm-rewriteRule "(.+,)?<realnamingcontext>$" "$1<virtualnamingcontext>" ":"
rwm-rewriteContext searchAttrDN alias searchEntryDN
rwm-rewriteContext matchedDN alias searchEntryDN
# misc empty rules
rwm-rewriteContext referralAttrDN
rwm-rewriteContext referralDN

# Everything defined here goes into the 'default' context.
# This rule changes the naming context of anything sent
# to 'dc=home,dc=net' to 'dc=OpenLDAP, dc=org'
```

```
rwm-rewriteRule "(.+,)?dc=home,[ ]?dc=net$"
        "$1dc=OpenLDAP, dc=org"  ":"

# since a pretty/normalized DN does not include spaces
# after rdn separators, e.g. ‘,’, this rule suffices:

rwm-rewriteRule "(.+,)?dc=home,dc=net$"
        "$1dc=OpenLDAP,dc=org"  ":"

# Start a new context (ends input of the previous one).
# This rule adds blanks between DN parts if not present.
rwm-rewriteContext  addBlanks
rwm-rewriteRule     "(.*),([^ ].*)" "$1, $2"

# This one eats blanks
rwm-rewriteContext  eatBlanks
rwm-rewriteRule     "(.*), (.*)" "$1,$2"

# Here control goes back to the default rewrite
# context; rules are appended to the existing ones.
# anything that gets here is piped into rule ‘addBlanks’
rwm-rewriteContext  default
rwm-rewriteRule     ".*" "${>addBlanks($0)}" ":"

# Rewrite the search base according to ‘default’ rules.
rwm-rewriteContext  searchDN alias default

# Search results with OpenLDAP DN are rewritten back with
# ‘dc=home,dc=net’ naming context, with spaces eaten.
rwm-rewriteContext  searchEntryDN
rwm-rewriteRule     "(.*[^ ],)?[ ]?dc=OpenLDAP,[ ]?dc=org$"
            "${>eatBlanks($1)}dc=home,dc=net"    ":"

# Bind with email instead of full DN: we first need
# an ldap map that turns attributes into a DN (the
# argument used when invoking the map is appended to
# the URI and acts as the filter portion)
rwm-rewriteMap ldap attr2dn "ldap://host/dc=my,dc=org?dn?sub"

# Then we need to detect DN made up of a single email,
# e.g. ‘mail=someone@example.com’; note that the rule
# in case of match stops rewriting; in case of error,
# it is ignored.  In case we are mapping virtual
# to real naming contexts, we also need to rewrite
# regular DNs, because the definition of a bindDN
# rewrite context overrides the default definition.
rwm-rewriteContext bindDN
rwm-rewriteRule "^mail=[^,]+@[^,]+$" "${attr2dn($0)}" ":@I"

# This is a rather sophisticated example. It massages a
# search filter in case who performs the search has
# administrative privileges.  First we need to keep
# track of the bind DN of the incoming request, which is
# stored in a variable called ‘binddn’ with session scope,
```

```
# and left in place to allow regular binding:
rwm-rewriteContext  bindDN
rwm-rewriteRule     ".+" "${&&binddn($0)}$0" ":"

# A search filter containing ‘uid=’ is rewritten only
# if an appropriate DN is bound.
# To do this, in the first rule the bound DN is
# dereferenced, while the filter is decomposed in a
# prefix, in the value of the ‘uid=<arg>’ AVA, and
# in a suffix. A tag ‘<>’ is appended to the DN.
# If the DN refers to an entry in the ‘ou=admin’ subtree,
# the filter is rewritten OR-ing the ‘uid=<arg>’ with
# ‘cn=<arg>’; otherwise it is left as is. This could be
# useful, for instance, to allow apache’s auth_ldap-1.4
# module to authenticate users with both ‘uid’ and
# ‘cn’, but only if the request comes from a possible
# ‘cn=Web auth,ou=admin,dc=home,dc=net’ user.
rwm-rewriteContext searchFilter
rwm-rewriteRule "(.*\\()uid=([a-z0-9_]+)(\\).*)"
  "${**binddn}<>${&prefix($1)}${&arg($2)}${&suffix($3)}"
  ":I"
rwm-rewriteRule "^[^,]+,ou=admin,dc=home,dc=net$"
  "${*prefix}|(uid=${*arg})(cn=${*arg})${*suffix}" ":@I"
rwm-rewriteRule ".*<>$" "${*prefix}uid=${*arg}${*suffix}" ":"

# This example shows how to strip unwanted DN-valued
# attribute values from a search result; the first rule
# matches DN values below "ou=People,dc=example,dc=com";
# in case of match the rewriting exits successfully.
# The second rule matches everything else and causes
# the value to be rejected.
rwm-rewriteContext searchEntryDN
rwm-rewriteRule ".+,ou=People,dc=example,dc=com$" "$0" ":@"
rwm-rewriteRule ".*" "" "#"
```

MAPPING EXAMPLES

The following directives map the object class ‘groupOfNames’ to the object class ‘groupOfUniqueNames’ and the attribute type ‘member’ to the attribute type ‘uniqueMember’:

```
map objectclass groupOfNames groupOfUniqueNames
map attribute uniqueMember member
```

This presents a limited attribute set from the foreign server:

```
map attribute cn *
map attribute sn *
map attribute manager *
map attribute description *
map attribute *
```

These lines map cn, sn, manager, and description to themselves, and any other attribute gets "removed" from the object before it is sent to the client (or sent up to the LDAP server). This is obviously a simplistic example, but you get the point.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapd−meta**(5), **slapd−relay**(5), **slapd**(8), **regex**(7), **re_format**(7).

AUTHOR

Pierangelo Masarati; based on back-ldap rewrite/remap features by Howard Chu, Pierangelo Masarati.

NAME

slapo-rwm − rewrite/remap overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The **rwm** overlay to **slapd**(8) performs basic DN/data rewrite and objectClass/attributeType mapping. Its usage is mostly intended to provide virtual views of existing data either remotely, in conjunction with the proxy backend described in **slapd-ldap**(5), or locally, in conjunction with the relay backend described in **slapd-relay**(5).

This overlay is experimental.

MAPPING

An important feature of the **rwm** overlay is the capability to map objectClasses and attributeTypes from the local set (or a subset of it) to a foreign set, and vice versa. This is accomplished by means of the **rwm-map** directive.

rwm-map {attribute | objectclass} [<local name> | *] {<foreign name> | *}

Map attributeTypes and objectClasses from the foreign server to different values on the local slapd. The reason is that some attributes might not be part of the local slapd's schema, some attribute names might be different but serve the same purpose, etc. If local or foreign name is '*', the name is preserved. If local name is omitted, the foreign name is removed. Unmapped names are preserved if both local and foreign name are '*', and removed if local name is omitted and foreign name is '*'.

The local *objectClasses* and *attributeTypes* must be defined in the local schema; the foreign ones do not have to, but users are encouraged to explicitly define the remote attributeTypes and the objectClasses they intend to map. All in all, when remapping a remote server via back-ldap (**slapd-ldap**(5)) or back-meta (**slapd-meta**(5)) their definition can be easily obtained by querying the *subschemaSubentry* of the remote server; the problem should not exist when remapping a local database. Note, however, that the decision whether to rewrite or not attributeTypes with *distinguishedName syntax*, requires the knowledge of the attributeType syntax. See the REWRITING section for details.

Note that when mapping DN-valued attributes from local to remote, first the DN is rewritten, and then the attributeType is mapped; while mapping from remote to local, first the attributeType is mapped, and then the DN is rewritten. As such, it is important that the local attributeType is appropriately defined as using the distinguishedName syntax. Also, note that there are DN-related syntaxes (i.e. compound types with a portion that is DN-valued), like nameAndOptionalUID, whose values are currently not rewritten.

If the foreign type of an attribute mapping is not defined on the local server, it might be desirable to have the attribute values normalized after the mapping process. Not normalizing the values can lead to wrong results, when the **rwm** overlay is used together with e.g. the **pcache** overlay. This normalization can be enabled by means of the **rwm-normalize-mapped-attrs** directive.

rwm-normalize-mapped-attrs {yes|no}

Set this to "yes", if the **rwm** overlay should try to normalize the values of attributes that are mapped from an attribute type that is unknown to the local server. The default value of this setting is "no".

SUFFIX MASSAGING

A basic feature of the **rwm** overlay is the capability to perform suffix massaging between a virtual and a real naming context by means of the **rwm-suffixmassage** directive. This, in conjunction with proxy backends, **slapd-ldap**(5) and **slapd-meta**(5), or with the relay backend, **slapd-relay**(5), allows to create virtual views of databases. A distinguishing feature of this overlay is that, when instantiated before any database, it can modify the DN of requests *before* database selection. For this reason, rules that rewrite the empty DN ("") or the subschemaSubentry DN (usually "cn=subschema"), would prevent clients from reading the root DSE or the DSA's schema.

rwm-suffixmassage [<virtual naming context>] <real naming context>

Shortcut to implement naming context rewriting; the trailing part of the DN is rewritten from the virtual to the real naming context in the bindDN, searchDN, searchFilterAttrDN, compareDN, compareAttrDN, addDN, addAttrDN, modifyDN, modifyAttrDN, modrDN, newSuperiorDN, deleteDN, exopPasswdDN, and from the real to the virtual naming context in the searchEntryDN, searchAttrDN and matchedDN rewrite contexts. By default no rewriting occurs for the searchFilter and for the referralAttrDN and referralDN rewrite contexts. If no *<virtual naming context>* is given, the first suffix of the database is used; this requires the **rwm-suffixmassage** directive be defined *after* the database **suffix** directive. The **rwm-suffixmassage** directive automatically sets the **rwm-rewriteEngine** to **ON**.

See the REWRITING section for details.

REWRITING

A string is rewritten according to a set of rules, called a 'rewrite context'. The rules are based on POSIX (''extended'') regular expressions with substring matching; basic variable substitution and map resolution of substrings is allowed by specific mechanisms detailed in the following. The behavior of pattern matching/substitution can be altered by a set of flags.

<rewrite context> ::= <rewrite rule> [...]
<rewrite rule> ::= <pattern> <action> [<flags>]

The underlying concept is to build a lightweight rewrite module for the slapd server (initially dedicated to the LDAP backend):

Passes

An incoming string is matched against a set of *rewriteRules*. Rules are made of a *regex match pattern*, a *substitution pattern* and a set of actions, described by a set of *optional flags*. In case of match, string rewriting is performed according to the substitution pattern that allows to refer to substrings matched in the incoming string. The actions, if any, are finally performed. Each rule is executed recursively, unless altered by specific action flags; see "Action Flags" for details. A default limit on the recursion level is set, and can be altered by the **rwm-rewriteMaxPasses** directive, as detailed in the "Additional Configuration Syntax" section. The substitution pattern allows map resolution of substrings. A map is a generic object that maps a substitution pattern to a value. The flags are divided in "Pattern Matching Flags" and "Action Flags"; the former alter the regex match pattern behavior, while the latter alter the actions that are taken after substitution.

Pattern Matching Flags

'C' honors case in matching (default is case insensitive)

'R' use POSIX ''basic'' regular expressions (default is ''extended'')

'M{n}' allow no more than **n** recursive passes for a specific rule; does not alter the max total count of passes, so it can only enforce a stricter limit for a specific rule.

Action Flags

':' apply the rule once only (default is recursive)

'@' stop applying rules in case of match; the current rule is still applied recursively; combine with ':' to apply the current rule only once and then stop.

'#' stop current operation if the rule matches, and issue an 'unwilling to perform' error.

'G{n}' jump **n** rules back and forth (watch for loops!). Note that 'G{1}' is implicit in every rule.

'I' ignores errors in rule; this means, in case of error, e.g. issued by a map, the error is treated as a missed match. The 'unwilling to perform' is not overridden.

'U{n}' uses **n** as return code if the rule matches; the flag does not alter the recursive behavior of the rule, so, to have it performed only once, it must be used in combination with ':', e.g. **':U{32}'** returns the value '32' (indicating noSuchObject) after exactly one execution of the rule, if the pattern matches. As a consequence, its behavior is equivalent to '@', with the return code set to **n**; or, in

other words, '@' is equivalent to 'U{0}'. Positive errors are allowed, indicating the related LDAP error codes as specified in *draft-ietf-ldapbis-protocol*.

The ordering of the flags can be significant. For instance: 'IG{2}' means ignore errors and jump two lines ahead both in case of match and in case of error, while 'G{2}I' means ignore errors, but jump two lines ahead only in case of match.

More flags (mainly Action Flags) will be added as needed.

Pattern Matching

See **regex**(7) and/or **re_format**(7).

Substitution Pattern Syntax

Everything starting with '$' requires substitution;

the only obvious exception is '$$', which is turned into a single '$';

the basic substitution is '$<d>', where '<d>' is a digit; 0 means the whole string, while 1-9 is a submatch, as discussed in **regex**(7) and/or **re_format**(7).

a '$' followed by a '{' invokes an advanced substitution. The pattern is:

'$' '{' [<operator>] <name> '(' <substitution> ')' '}'

where <name> must be a legal name for the map, i.e.

<name> ::= [a-z][a-z0-9]* (case insensitive)
<operator> ::= '>' '|' '&' '&&' '*' '**' '$'

and <substitution> must be a legal substitution pattern, with no limits on the nesting level.

The operators are:

> sub-context invocation; <name> must be a legal, already defined rewrite context name

| external command invocation; <name> must refer to a legal, already defined command name (NOT IMPLEMENTED YET)

& variable assignment; <name> defines a variable in the running operation structure which can be dereferenced later; operator **&** assigns a variable in the rewrite context scope; operator **&&** assigns a variable that scopes the entire session, e.g. its value can be dereferenced later by other rewrite contexts

***** variable dereferencing; <name> must refer to a variable that is defined and assigned for the running operation; operator ***** dereferences a variable scoping the rewrite context; operator ****** dereferences a variable scoping the whole session, e.g. the value is passed across rewrite contexts

$ parameter dereferencing; <name> must refer to an existing parameter; the idea is to make some run-time parameters set by the system available to the rewrite engine, as the client host name, the bind DN if any, constant parameters initialized at config time, and so on; no parameter is currently set by either **back−ldap** or **back−meta**, but constant parameters can be defined in the configuration file by using the **rewriteParam** directive.

Substitution escaping has been delegated to the '$' symbol, which is used instead of '\' in string substitution patterns because '\' is already escaped by slapd's low level parsing routines; as a consequence, regex escaping requires two '\' symbols, e.g. '**.*\.foo\.bar**' must be written as '**.*\\.foo\\.bar**'.

Rewrite Context

A rewrite context is a set of rules which are applied in sequence. The basic idea is to have an application initialize a rewrite engine (think of Apache's mod_rewrite ...) with a set of rewrite contexts; when string rewriting is required, one invokes the appropriate rewrite context with the input string and obtains the newly rewritten one if no errors occur.

Each basic server operation is associated to a rewrite context; they are divided in two main groups: client −> server and server −> client rewriting.

client -> server:

(default) if defined and no specific context is available
bindDN bind
searchDN search
searchFilter search
searchFilterAttrDN search
compareDN compare
compareAttrDN compare AVA
addDN add
addAttrDN add AVA (DN portion of "ref" excluded)
modifyDN modify
modifyAttrDN modify AVA (DN portion of "ref" excluded)
referralAttrDN add/modify DN portion of referrals (default to none)
modrDN modrdn
newSuperiorDN modrdn
deleteDN delete
exopPasswdDN password modify extended operation DN

server -> client:

searchEntryDN search (only if defined; no default; acts on DN of search entries)
searchAttrDN search AVA (only if defined; defaults to searchEntryDN; acts on DN-syntax attributes of search results)
matchedDN all ops (only if applicable; defaults to searchEntryDN)
referralDN all ops (only if applicable; defaults to none)

Basic Configuration Syntax

All rewrite/remap directives start with the prefix **rwm-**; for backwards compatibility with the historical **slapd-ldap**(5) and **slapd-meta**(5) builtin rewrite/remap capabilities, the prefix may be omitted, but this practice is strongly discouraged.

rwm-rewriteEngine { on | off }

If 'on', the requested rewriting is performed; if 'off', no rewriting takes place (an easy way to stop rewriting without altering too much the configuration file).

rwm-rewriteContext <context name> [alias <aliased context name>]

<Context name> is the name that identifies the context, i.e. the name used by the application to refer to the set of rules it contains. It is used also to reference sub contexts in string rewriting. A context may alias another one. In this case the alias context contains no rule, and any reference to it will result in accessing the aliased one.

rwm-rewriteRule <regex match pattern> <substitution pattern> [<flags>]

Determines how a string can be rewritten if a pattern is matched. Examples are reported below.

Additional Configuration Syntax

rwm-rewriteMap <map type> <map name> [<map attrs>]

Allows to define a map that transforms substring rewriting into something else. The map is referenced inside the substitution pattern of a rule.

rwm-rewriteParam <param name> <param value>

Sets a value with global scope, that can be dereferenced by the command '${$paramName}'.

rwm-rewriteMaxPasses <number of passes> [<number of passes per rule>]
Sets the maximum number of total rewriting passes that can be performed in a single rewrite operation (to avoid loops). A safe default is set to 100; note that reaching this limit is still treated as a success; recursive invocation of rules is simply interrupted. The count applies to the rewriting operation as a whole, not to any single rule; an optional per-rule limit can be set. This limit is overridden by setting specific per-rule limits with the 'M{n}' flag.

MAPS

Currently, few maps are builtin and there are no provisions for developers to register new map types at runtime.

Supported maps are:

LDAP <URI> [bindwhen=<when>] [version=<version>] [binddn=<DN>] [credentials=<cred>]
The **LDAP** map expands a value by performing a simple LDAP search. Its configuration is based on a mandatory URI, whose **attrs** portion must contain exactly one attribute (use **entryDN** to fetch the DN of an entry). If a multi-valued attribute is used, only the first value is considered.

The parameter **bindwhen** determines when the connection is established. It can take the values **now**, **later**, and **everytime**, respectively indicating that the connection should be created at startup, when required, or any time it is used. In the former two cases, the connection is cached, while in the latter a fresh new one is used all times. This is the default.

The parameters **binddn** and **credentials** represent the DN and the password that is used to perform an authenticated simple bind before performing the search operation; if not given, an anonymous connection is used.

The parameter **version** can be 2 or 3 to indicate the protocol version that must be used. The default is 3.

REWRITE CONFIGURATION EXAMPLES

```
# set to 'off' to disable rewriting
rwm-rewriteEngine on

# the rules the "suffixmassage" directive implies
rwm-rewriteEngine on
# all dataflow from client to server referring to DNs
rwm-rewriteContext default
rwm-rewriteRule "(.+,)?<virtualnamingcontext>$" "$1<realnamingcontext>" ":"
# empty filter rule
rwm-rewriteContext searchFilter
# all dataflow from server to client
rwm-rewriteContext searchEntryDN
rwm-rewriteRule "(.+,)?<realnamingcontext>$" "$1<virtualnamingcontext>" ":"
rwm-rewriteContext searchAttrDN alias searchEntryDN
rwm-rewriteContext matchedDN alias searchEntryDN
# misc empty rules
rwm-rewriteContext referralAttrDN
rwm-rewriteContext referralDN

# Everything defined here goes into the 'default' context.
# This rule changes the naming context of anything sent
# to 'dc=home,dc=net' to 'dc=OpenLDAP, dc=org'
```

```
rwm-rewriteRule "(.+,)?dc=home,[ ]?dc=net$"
        "$1dc=OpenLDAP, dc=org"  ":"

# since a pretty/normalized DN does not include spaces
# after rdn separators, e.g. ‘,’, this rule suffices:

rwm-rewriteRule "(.+,)?dc=home,dc=net$"
        "$1dc=OpenLDAP,dc=org"  ":"

# Start a new context (ends input of the previous one).
# This rule adds blanks between DN parts if not present.
rwm-rewriteContext  addBlanks
rwm-rewriteRule     "(.*),([^ ].*)" "$1, $2"

# This one eats blanks
rwm-rewriteContext  eatBlanks
rwm-rewriteRule     "(.*), (.*)" "$1,$2"

# Here control goes back to the default rewrite
# context; rules are appended to the existing ones.
# anything that gets here is piped into rule ‘addBlanks’
rwm-rewriteContext  default
rwm-rewriteRule     ".*" "${>addBlanks($0)}" ":"

# Rewrite the search base according to ‘default’ rules.
rwm-rewriteContext  searchDN alias default

# Search results with OpenLDAP DN are rewritten back with
# ‘dc=home,dc=net’ naming context, with spaces eaten.
rwm-rewriteContext  searchEntryDN
rwm-rewriteRule     "(.*[^ ],)?[ ]?dc=OpenLDAP,[ ]?dc=org$"
            "${>eatBlanks($1)}dc=home,dc=net"    ":"

# Bind with email instead of full DN: we first need
# an ldap map that turns attributes into a DN (the
# argument used when invoking the map is appended to
# the URI and acts as the filter portion)
rwm-rewriteMap ldap attr2dn "ldap://host/dc=my,dc=org?dn?sub"

# Then we need to detect DN made up of a single email,
# e.g. ‘mail=someone@example.com’; note that the rule
# in case of match stops rewriting; in case of error,
# it is ignored.  In case we are mapping virtual
# to real naming contexts, we also need to rewrite
# regular DNs, because the definition of a bindDN
# rewrite context overrides the default definition.
rwm-rewriteContext bindDN
rwm-rewriteRule "^mail=[^,]+@[^,]+$" "${attr2dn($0)}" ":@I"

# This is a rather sophisticated example. It massages a
# search filter in case who performs the search has
# administrative privileges.  First we need to keep
# track of the bind DN of the incoming request, which is
# stored in a variable called ‘binddn’ with session scope,
```

```
# and left in place to allow regular binding:
rwm-rewriteContext  bindDN
rwm-rewriteRule     ".+" "${&&binddn($0)}$0" ":"

# A search filter containing ‘uid=’ is rewritten only
# if an appropriate DN is bound.
# To do this, in the first rule the bound DN is
# dereferenced, while the filter is decomposed in a
# prefix, in the value of the ‘uid=<arg>’ AVA, and
# in a suffix. A tag ‘<>’ is appended to the DN.
# If the DN refers to an entry in the ‘ou=admin’ subtree,
# the filter is rewritten OR-ing the ‘uid=<arg>’ with
# ‘cn=<arg>’; otherwise it is left as is. This could be
# useful, for instance, to allow apache’s auth_ldap-1.4
# module to authenticate users with both ‘uid’ and
# ‘cn’, but only if the request comes from a possible
# ‘cn=Web auth,ou=admin,dc=home,dc=net’ user.
rwm-rewriteContext searchFilter
rwm-rewriteRule "(.*\\()uid=([a-z0-9_]+)(\\).*)"
  "${**binddn}<>${&prefix($1)}${&arg($2)}${&suffix($3)}"
  ":I"
rwm-rewriteRule "^[^,]+,ou=admin,dc=home,dc=net$"
  "${*prefix}|(uid=${*arg})(cn=${*arg})${*suffix}" ":@I"
rwm-rewriteRule ".*<>$" "${*prefix}uid=${*arg}${*suffix}" ":"

# This example shows how to strip unwanted DN-valued
# attribute values from a search result; the first rule
# matches DN values below "ou=People,dc=example,dc=com";
# in case of match the rewriting exits successfully.
# The second rule matches everything else and causes
# the value to be rejected.
rwm-rewriteContext searchEntryDN
rwm-rewriteRule ".+,ou=People,dc=example,dc=com$" "$0" ":@"
rwm-rewriteRule ".*" "" "#"
```

MAPPING EXAMPLES

The following directives map the object class ‘groupOfNames’ to the object class ‘groupOfUniqueNames’ and the attribute type ‘member’ to the attribute type ‘uniqueMember’:

```
map objectclass groupOfNames groupOfUniqueNames
map attribute uniqueMember member
```

This presents a limited attribute set from the foreign server:

```
map attribute cn *
map attribute sn *
map attribute manager *
map attribute description *
map attribute *
```

These lines map cn, sn, manager, and description to themselves, and any other attribute gets "removed" from the object before it is sent to the client (or sent up to the LDAP server). This is obviously a simplistic example, but you get the point.

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd−ldap**(5), **slapd−meta**(5), **slapd−relay**(5), **slapd**(8), **regex**(7), **re_format**(7).

AUTHOR

Pierangelo Masarati; based on back-ldap rewrite/remap features by Howard Chu, Pierangelo Masarati.

NAME

slapo-syncprov − Sync Provider overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Sync Provider overlay implements the provider-side support for the LDAP Content Synchronization (RFC4533) as well as syncrepl replication support. The overlay can be used with any backend that maintains entryCSN and entryUUID attributes for its entries. It also creates a contextCSN attribute in the root entry of the database.

The contextCSN is updated for every write operation performed against the database. To reduce database contention, the contextCSN is only updated in memory. The value is written to the database on server shutdown and read into memory on startup, and maintained in memory thereafter. Checkpoints may be configured to write the contextCSN into the underlying database to minimize recovery time after an unclean shutdown.

On databases that support inequality indexing, it is helpful to set an eq index on the entryCSN attribute when using this overlay.

CONFIGURATION

These **slapd.conf** options apply to the Sync Provider overlay. They should appear after the **overlay** directive.

syncprov-checkpoint <ops> <minutes>

After a write operation has succeeded, write the contextCSN to the underlying database if **<ops>** write operations or more than **<minutes>** time have passed since the last checkpoint. Checkpointing is disabled by default.

syncprov-sessionlog <ops>

Specify a session log for recording information about write operations made on the database. The **<ops>** specifies the number of operations that are recorded in the log. All write operations (except Adds) are recorded in the log. When using the session log, it is helpful to set an eq index on the entryUUID attribute in the underlying database.

syncprov-nopresent TRUE | FALSE

Specify that the Present phase of refreshing should be skipped. This value should only be set TRUE for a syncprov instance on top of a log database (such as one managed by the accesslog overlay). The default is FALSE.

syncprov-reloadhint TRUE | FALSE

Specify that the overlay should honor the reloadHint flag in the Sync Control. In OpenLDAP releases 2.3.11 and earlier the syncrepl consumer did not properly set this flag, so the overlay must ignore it. This option should be set TRUE when working with newer releases that properly support this flag. It must be set TRUE when using the accesslog overlay for delta-based syncrepl replication support. The default is FALSE.

FILES

ETCDIR/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapo-accesslog**(5). OpenLDAP Administrator's Guide.

ACKNOWLEDGEMENTS

NAME

slapo-syncprov − Sync Provider overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Sync Provider overlay implements the provider-side support for the LDAP Content Synchronization (RFC4533) as well as syncrepl replication support. The overlay can be used with any backend that maintains entryCSN and entryUUID attributes for its entries. It also creates a contextCSN attribute in the root entry of the database.

The contextCSN is updated for every write operation performed against the database. To reduce database contention, the contextCSN is only updated in memory. The value is written to the database on server shutdown and read into memory on startup, and maintained in memory thereafter. Checkpoints may be configured to write the contextCSN into the underlying database to minimize recovery time after an unclean shutdown.

On databases that support inequality indexing, it is helpful to set an eq index on the entryCSN attribute when using this overlay.

CONFIGURATION

These **slapd.conf** options apply to the Sync Provider overlay. They should appear after the **overlay** directive.

syncprov-checkpoint <ops> <minutes>

After a write operation has succeeded, write the contextCSN to the underlying database if **<ops>** write operations or more than **<minutes>** time have passed since the last checkpoint. Checkpointing is disabled by default.

syncprov-sessionlog <ops>

Specify a session log for recording information about write operations made on the database. The **<ops>** specifies the number of operations that are recorded in the log. All write operations (except Adds) are recorded in the log. When using the session log, it is helpful to set an eq index on the entryUUID attribute in the underlying database.

syncprov-nopresent TRUE | FALSE

Specify that the Present phase of refreshing should be skipped. This value should only be set TRUE for a syncprov instance on top of a log database (such as one managed by the accesslog overlay). The default is FALSE.

syncprov-reloadhint TRUE | FALSE

Specify that the overlay should honor the reloadHint flag in the Sync Control. In OpenLDAP releases 2.3.11 and earlier the syncrepl consumer did not properly set this flag, so the overlay must ignore it. This option should be set TRUE when working with newer releases that properly support this flag. It must be set TRUE when using the accesslog overlay for delta-based syncrepl replication support. The default is FALSE.

FILES

/etc/openldap/slapd.conf

default slapd configuration file

SEE ALSO

slapd.conf(5), **slapo-accesslog**(5). OpenLDAP Administrator's Guide.

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from University of Michigan LDAP 3.3 Release.

NAME

slapo-translucent − Translucent Proxy overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Translucent Proxy overlay can be used with a backend database such as **slapd-bdb**(5) to create a "translucent proxy". Entries retrieved from a remote LDAP server may have some or all attributes overridden, or new attributes added, by entries in the local database before being presented to the client.

A **search** operation is first populated with entries from the remote LDAP server, the attributes of which are then overridden with any attributes defined in the local database. Local overrides may be populated with the **add**, **modify ,** and **modrdn** operations, the use of which is restricted to the root user.

A **compare** operation will perform a comparison with attributes defined in the local database record (if any) before any comparison is made with data in the remote database.

CONFIGURATION

The Translucent Proxy overlay uses a remote LDAP server which is configured with the options shown in **slapd-ldap**(5). These **slapd.conf** options are specific to the Translucent Proxy overlay; they must appear after the **overlay** directive.

translucent_strict
: By default, attempts to delete attributes in either the local or remote databases will be silently ignored. The **translucent_strict** directive causes these modifications to fail with a Constraint Violation.

translucent_no_glue
: This configuration option disables the automatic creation of "glue" records for an **add** or **modrdn** operation, such that all parents of an entry added to the local database must be created by hand. Glue records are always created for a **modify** operation.

translucent_local <attr[,attr...]>
: Specify a list of attributes that should be searched for in the local database when used in a search filter. By default, search filters are only handled by the remote database. With this directive, search filters will be split into a local and remote portion, and local attributes will be searched locally.

translucent_remote <attr[,attr...]>
: Specify a list of attributes that should be searched for in the remote database when used in a search filter. This directive complements the **translucent_local** directive. Attributes may be specified as both local and remote if desired.

If neither **translucent_local** nor **translucent_remote** are specified, the default behavior is to search the remote database with the complete search filter. If only **translucent_local** is specified, searches will only be run on the local database. Likewise, if only **translucent_remote** is specified, searches will only be run on the remote database. In any case, both the local and remote entries corresponding to a search result will be merged before being returned to the client.

CAVEATS

The Translucent Proxy overlay will disable schema checking in the local database, so that an entry consisting of overlay attributes need not adhere to the complete schema.

Because the translucent overlay does not perform any DN rewrites, the local and remote database instances must have the same suffix. Other configurations will probably fail with No Such Object and other errors.

FILES

ETCDIR/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd-ldap**(5).

NAME

slapo-translucent − Translucent Proxy overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Translucent Proxy overlay can be used with a backend database such as **slapd-bdb**(5) to create a "translucent proxy". Entries retrieved from a remote LDAP server may have some or all attributes overridden, or new attributes added, by entries in the local database before being presented to the client.

A **search** operation is first populated with entries from the remote LDAP server, the attributes of which are then overridden with any attributes defined in the local database. Local overrides may be populated with the **add**, **modify ,** and **modrdn** operations, the use of which is restricted to the root user.

A **compare** operation will perform a comparison with attributes defined in the local database record (if any) before any comparison is made with data in the remote database.

CONFIGURATION

The Translucent Proxy overlay uses a remote LDAP server which is configured with the options shown in **slapd-ldap**(5). These **slapd.conf** options are specific to the Translucent Proxy overlay; they must appear after the **overlay** directive.

translucent_strict
: By default, attempts to delete attributes in either the local or remote databases will be silently ignored. The **translucent_strict** directive causes these modifications to fail with a Constraint Violation.

translucent_no_glue
: This configuration option disables the automatic creation of "glue" records for an **add** or **modrdn** operation, such that all parents of an entry added to the local database must be created by hand. Glue records are always created for a **modify** operation.

translucent_local <attr[,attr...]>
: Specify a list of attributes that should be searched for in the local database when used in a search filter. By default, search filters are only handled by the remote database. With this directive, search filters will be split into a local and remote portion, and local attributes will be searched locally.

translucent_remote <attr[,attr...]>
: Specify a list of attributes that should be searched for in the remote database when used in a search filter. This directive complements the **translucent_local** directive. Attributes may be specified as both local and remote if desired.

If neither **translucent_local** nor **translucent_remote** are specified, the default behavior is to search the remote database with the complete search filter. If only **translucent_local** is specified, searches will only be run on the local database. Likewise, if only **translucent_remote** is specified, searches will only be run on the remote database. In any case, both the local and remote entries corresponding to a search result will be merged before being returned to the client.

CAVEATS

The Translucent Proxy overlay will disable schema checking in the local database, so that an entry consisting of overlay attributes need not adhere to the complete schema.

Because the translucent overlay does not perform any DN rewrites, the local and remote database instances must have the same suffix. Other configurations will probably fail with No Such Object and other errors.

FILES

/etc/openldap/slapd.conf
: default slapd configuration file

SEE ALSO

slapd.conf(5), **slapd-ldap**(5).

NAME

slapo-unique − Attribute Uniqueness overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Attribute Uniqueness overlay can be used with a backend database such as **slapd-bdb**(5) to enforce the uniqueness of some or all attributes within a scope. This subtree defaults to all objects within the subtree of the database for which the Uniqueness overlay is configured.

Uniqueness is enforced by searching the subtree to ensure that the values of all attributes presented with an **add**, **modify** or **modrdn** operation are unique within the scope. For example, if uniqueness were enforced for the **uid** attribute, the subtree would be searched for any other records which also have a **uid** attribute containing the same value. If any are found, the request is rejected.

CONFIGURATION

These **slapd.conf** options apply to the Attribute Uniqueness overlay. They should appear after the **overlay** directive.

unique_uri <[strict][ignore]URI[URI...]...>

Configure the base, attributes, scope, and filter for uniqueness checking. Multiple URIs may be specified within a domain, allowing complex selections of objects. Multiple **unique_uri** statements or **olcUniqueURI** attributes will create independent domains, each with their own independent lists of URIs and ignore/strict settings.

The LDAP URI syntax is a subset of **RFC-4516,** and takes the form:

ldap:///[base dn]?[attributes...]?scope[?filter]

The **base dn** defaults to that of the back-end database. Specified base dns must be within the subtree of the back-end database.

If no **attributes** are specified, the URI applies to all non-operational attributes.

The **scope** component is effectively mandatory, because LDAP URIs default to **base** scope, which is not valid for uniqueness, because groups of one object are always unique. Scopes of **sub** (for subtree) and **one** for one-level are valid.

The **filter** component causes the domain to apply uniqueness constraints only to matching objects. e.g. **ldap:///?cn?sub?(sn=e*)** would require unique **cn** attributes for all objects in the subtree of the back-end database whose **sn** starts with an e.

It is possible to assert uniqueness upon all non-operational attributes except those listed by prepending the keyword **ignore** If not configured, all non-operational (e.g., system) attributes must be unique. Note that the **attributes** list of an **ignore** URI should generally contain the **objectClass**, **dc**, **ou** and **o** attributes, as these will generally not be unique, nor are they operational attributes.

It is possible to set strict checking for the uniqueness domain by prepending the keyword **strict.** By default, uniqueness is not enforced for null values. Enabling **strict** mode extends the concept of uniqueness to include null values, such that only one attribute within a subtree will be allowed to have a null value. Strictness applies to all URIs within a uniqueness domain, but some domains may be strict while others are not.

It is not possible to set both URIs and legacy slapo-unique configuration parameters simultaneously. In general, the legacy configuration options control pieces of a single unfiltered subtree domain.

unique_base <basedn>
This legacy configuration parameter should be converted to the **base dn** component of the above **unique_uri** style of parameter.

unique_ignore <attribute...>
This legacy configuration parameter should be converted to a **unique_uri** parameter with **ignore** keyword as described above.

unique_attributes <attribute...>
This legacy configuration parameter should be converted to a **unique_uri** parameter, as described above.

unique_strict
This legacy configuration parameter should be converted to a **strict** keyword prepended to a **unique_uri** parameter, as described above.

CAVEATS

unique_uri cannot be used with the old-style of configuration, and vice versa. **unique_uri** can implement everything the older system can do, however.

Typical attributes for the **ignore ldap:///...** URIs are intentionally not hardcoded into the overlay to allow for maximum flexibility in meeting site-specific requirements.

FILES

ETCDIR/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5).

NAME

slapo-unique − Attribute Uniqueness overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Attribute Uniqueness overlay can be used with a backend database such as **slapd-bdb**(5) to enforce the uniqueness of some or all attributes within a scope. This subtree defaults to all objects within the subtree of the database for which the Uniqueness overlay is configured.

Uniqueness is enforced by searching the subtree to ensure that the values of all attributes presented with an **add**, **modify** or **modrdn** operation are unique within the scope. For example, if uniqueness were enforced for the **uid** attribute, the subtree would be searched for any other records which also have a **uid** attribute containing the same value. If any are found, the request is rejected.

CONFIGURATION

These **slapd.conf** options apply to the Attribute Uniqueness overlay. They should appear after the **overlay** directive.

unique_uri <[strict][ignore]URI[URI...]...>

Configure the base, attributes, scope, and filter for uniqueness checking. Multiple URIs may be specified within a domain, allowing complex selections of objects. Multiple **unique_uri** statements or **olcUniqueURI** attributes will create independent domains, each with their own independent lists of URIs and ignore/strict settings.

The LDAP URI syntax is a subset of **RFC-4516,** and takes the form:

ldap:///[base dn]?[attributes...]?scope[?filter]

The **base dn** defaults to that of the back-end database. Specified base dns must be within the subtree of the back-end database.

If no **attributes** are specified, the URI applies to all non-operational attributes.

The **scope** component is effectively mandatory, because LDAP URIs default to **base** scope, which is not valid for uniqueness, because groups of one object are always unique. Scopes of **sub** (for subtree) and **one** for one-level are valid.

The **filter** component causes the domain to apply uniqueness constraints only to matching objects. e.g. **ldap:///?cn?sub?(sn=e*)** would require unique **cn** attributes for all objects in the subtree of the back-end database whose **sn** starts with an e.

It is possible to assert uniqueness upon all non-operational attributes except those listed by prepending the keyword **ignore** If not configured, all non-operational (e.g., system) attributes must be unique. Note that the **attributes** list of an **ignore** URI should generally contain the **objectClass**, **dc**, **ou** and **o** attributes, as these will generally not be unique, nor are they operational attributes.

It is possible to set strict checking for the uniqueness domain by prepending the keyword **strict.** By default, uniqueness is not enforced for null values. Enabling **strict** mode extends the concept of uniqueness to include null values, such that only one attribute within a subtree will be allowed to have a null value. Strictness applies to all URIs within a uniqueness domain, but some domains may be strict while others are not.

It is not possible to set both URIs and legacy slapo-unique configuration parameters simultaneously. In general, the legacy configuration options control pieces of a single unfiltered subtree domain.

unique_base <basedn>
This legacy configuration parameter should be converted to the **base dn** component of the above **unique_uri** style of parameter.

unique_ignore <attribute...>
This legacy configuration parameter should be converted to a **unique_uri** parameter with **ignore** keyword as described above.

unique_attributes <attribute...>
This legacy configuration parameter should be converted to a **unique_uri** parameter, as described above.

unique_strict
This legacy configuration parameter should be converted to a **strict** keyword prepended to a **unique_uri** parameter, as described above.

CAVEATS

unique_uri cannot be used with the old-style of configuration, and vice versa. **unique_uri** can implement everything the older system can do, however.

Typical attributes for the **ignore ldap:///...** URIs are intentionally not hardcoded into the overlay to allow for maximum flexibility in meeting site-specific requirements.

FILES

/etc/openldap/slapd.conf
default slapd configuration file

SEE ALSO

slapd.conf(5).

NAME

slapo-valsort − Value Sorting overlay to slapd

SYNOPSIS

ETCDIR/slapd.conf

DESCRIPTION

The Value Sorting overlay can be used with a backend database to sort the values of specific multi-valued attributes within a subtree. The sorting occurs whenever the attributes are returned in a search response.

Sorting can be specified in ascending or descending order, using either numeric or alphanumeric sort methods. Additionally, a "weighted" sort can be specified, which uses a numeric weight prepended to the attribute values. The weighted sort is always performed in ascending order, but may be combined with the other methods for values that all have equal weights. The weight is specified by prepending an integer weight {<*weight*>} in front of each value of the attribute for which weighted sorting is desired. This weighting factor is stripped off and never returned in search results.

CONFIGURATION

These *slapd.conf* options apply to the Value Sorting overlay. They should appear after the **overlay** directive.

valsort-attr <*attribute*> <*baseDN*> (<*sort-method*> | weighted [<*sort-method*>])

Configure a sorting method for the specified *attribute* in the subtree rooted at *baseDN*. The *sort-method* may be one of **alpha-ascend**, **alpha-descend**, **numeric-ascend**, or **numeric-descend**. If the special **weighted** method is specified, a secondary *sort-method* may also be specified. It is an error to specify an alphanumeric *sort-method* for an attribute with Integer or NumericString syntax, and it is an error to specify a numeric *sort-method* for an attribute with a syntax other than Integer or NumericString.

EXAMPLES

```
database bdb
suffix dc=example,dc=com
...
overlay valsort
valsort-attr member ou=groups,dc=example,dc=com alpha-ascend
```

FILES

ETCDIR/slapd.conf

default **slapd** configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

This module was written in 2005 by Howard Chu of Symas Corporation. The work was sponsored by Stanford University.

NAME

slapo-valsort − Value Sorting overlay to slapd

SYNOPSIS

/etc/openldap/slapd.conf

DESCRIPTION

The Value Sorting overlay can be used with a backend database to sort the values of specific multi-valued attributes within a subtree. The sorting occurs whenever the attributes are returned in a search response.

Sorting can be specified in ascending or descending order, using either numeric or alphanumeric sort methods. Additionally, a "weighted" sort can be specified, which uses a numeric weight prepended to the attribute values. The weighted sort is always performed in ascending order, but may be combined with the other methods for values that all have equal weights. The weight is specified by prepending an integer weight {<*weight*>} in front of each value of the attribute for which weighted sorting is desired. This weighting factor is stripped off and never returned in search results.

CONFIGURATION

These *slapd.conf* options apply to the Value Sorting overlay. They should appear after the **overlay** directive.

valsort-attr <*attribute*> <*baseDN*> (<*sort-method*> | weighted [<*sort-method*>])

Configure a sorting method for the specified *attribute* in the subtree rooted at *baseDN*. The *sort-method* may be one of **alpha-ascend**, **alpha-descend**, **numeric-ascend**, or **numeric-descend**. If the special **weighted** method is specified, a secondary *sort-method* may also be specified. It is an error to specify an alphanumeric *sort-method* for an attribute with Integer or NumericString syntax, and it is an error to specify a numeric *sort-method* for an attribute with a syntax other than Integer or NumericString.

EXAMPLES

```
database bdb
suffix dc=example,dc=com
...
overlay valsort
valsort-attr member ou=groups,dc=example,dc=com alpha-ascend
```

FILES

/etc/openldap/slapd.conf

default **slapd** configuration file

SEE ALSO

slapd.conf(5).

ACKNOWLEDGEMENTS

This module was written in 2005 by Howard Chu of Symas Corporation. The work was sponsored by Stanford University.

NAME

ssh_config — OpenSSH SSH client configuration files

SYNOPSIS

~/.ssh/config
/etc/ssh/ssh_config

DESCRIPTION

ssh(1) obtains configuration data from the following sources in the following order:

1. command-line options
2. user's configuration file (~/.ssh/config)
3. system-wide configuration file (/etc/ssh/ssh_config)

For each parameter, the first obtained value will be used. The configuration files contain sections separated by "Host" specifications, and that section is only applied for hosts that match one of the patterns given in the specification. The matched host name is the one given on the command line.

Since the first obtained value for each parameter is used, more host-specific declarations should be given near the beginning of the file, and general defaults at the end.

The configuration file has the following format:

Empty lines and lines starting with '#' are comments. Otherwise a line is of the format "keyword arguments". Configuration options may be separated by whitespace or optional whitespace and exactly one '='; the latter format is useful to avoid the need to quote whitespace when specifying configuration options using the **ssh**, **scp**, and **sftp -o** option. Arguments may optionally be enclosed in double quotes (") in order to represent arguments containing spaces.

The possible keywords and their meanings are as follows (note that keywords are case-insensitive and arguments are case-sensitive):

Host Restricts the following declarations (up to the next **Host** keyword) to be only for those hosts that match one of the patterns given after the keyword. A single '*' as a pattern can be used to provide global defaults for all hosts. The host is the *hostname* argument given on the command line (i.e. the name is not converted to a canonicalized host name before matching).

See **PATTERNS** for more information on patterns.

AddressFamily
Specifies which address family to use when connecting. Valid arguments are "any", "inet" (use IPv4 only), or "inet6" (use IPv6 only).

BatchMode
If set to "yes", passphrase/password querying will be disabled. This option is useful in scripts and other batch jobs where no user is present to supply the password. The argument must be "yes" or "no". The default is "no".

BindAddress
Use the specified address on the local machine as the source address of the connection. Only useful on systems with more than one address. Note that this option does not work if **UsePrivilegedPort** is set to "yes".

ChallengeResponseAuthentication
Specifies whether to use challenge-response authentication. The argument to this keyword must be "yes" or "no". The default is "yes".

CheckHostIP
If this flag is set to “yes”, ssh(1) will additionally check the host IP address in the known_hosts file. This allows ssh to detect if a host key changed due to DNS spoofing. If the option is set to “no”, the check will not be executed. The default is “yes”.

Cipher
Specifies the cipher to use for encrypting the session in protocol version 1. Currently, “blowfish”, “3des”, and “des” are supported. *des* is only supported in the ssh(1) client for interoperability with legacy protocol 1 implementations that do not support the *3des* cipher. Its use is strongly discouraged due to cryptographic weaknesses. The default is “3des”.

Ciphers
Specifies the ciphers allowed for protocol version 2 in order of preference. Multiple ciphers must be comma-separated. The supported ciphers are “3des-cbc”, “aes128-cbc”, “aes192-cbc”, “aes256-cbc”, “aes128-ctr”, “aes192-ctr”, “aes256-ctr”, “arcfour128”, “arcfour256”, “arcfour”, “blowfish-cbc”, and “cast128-cbc”. The default is:

```
aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour128,
arcfour256,arcfour,aes192-cbc,aes256-cbc,aes128-ctr,
aes192-ctr,aes256-ctr
```

ClearAllForwardings
Specifies that all local, remote, and dynamic port forwardings specified in the configuration files or on the command line be cleared. This option is primarily useful when used from the ssh(1) command line to clear port forwardings set in configuration files, and is automatically set by scp(1) and sftp(1). The argument must be “yes” or “no”. The default is “no”.

Compression
Specifies whether to use compression. The argument must be “yes” or “no”. The default is “no”.

CompressionLevel
Specifies the compression level to use if compression is enabled. The argument must be an integer from 1 (fast) to 9 (slow, best). The default level is 6, which is good for most applications. The meaning of the values is the same as in gzip(1). Note that this option applies to protocol version 1 only.

ConnectionAttempts
Specifies the number of tries (one per second) to make before exiting. The argument must be an integer. This may be useful in scripts if the connection sometimes fails. The default is 1.

ConnectTimeout
Specifies the timeout (in seconds) used when connecting to the SSH server, instead of using the default system TCP timeout. This value is used only when the target is down or really unreachable, not when it refuses the connection.

ControlMaster
Enables the sharing of multiple sessions over a single network connection. When set to “yes”, ssh(1) will listen for connections on a control socket specified using the **ControlPath** argument. Additional sessions can connect to this socket using the same **ControlPath** with **ControlMaster** set to “no” (the default). These sessions will try to reuse the master instance’s network connection rather than initiating new ones, but will fall back to connecting normally if the control socket does not exist, or is not listening.

Setting this to “ask” will cause ssh to listen for control connections, but require confirmation using the SSH_ASKPASS program before they are accepted (see ssh-add(1) for details). If the **ControlPath** cannot be opened, ssh will continue without connecting to a master instance.

X11 and ssh-agent(1) forwarding is supported over these multiplexed connections, however the display and agent forwarded will be the one belonging to the master connection i.e. it is not possible to forward multiple displays or agents.

Two additional options allow for opportunistic multiplexing: try to use a master connection but fall back to creating a new one if one does not already exist. These options are: “auto” and “autoask”. The latter requires confirmation like the “ask” option.

ControlPath

Specify the path to the control socket used for connection sharing as described in the **ControlMaster** section above or the string “none” to disable connection sharing. In the path, ‘%l’ will be substituted by the local host name, ‘%h’ will be substituted by the target host name, ‘%p’ the port, and ‘%r’ by the remote login username. It is recommended that any **ControlPath** used for opportunistic connection sharing include at least %h, %p, and %r. This ensures that shared connections are uniquely identified.

DynamicForward

Specifies that a TCP port on the local machine be forwarded over the secure channel, and the application protocol is then used to determine where to connect to from the remote machine.

The argument must be [*bind_address*:]*port*. IPv6 addresses can be specified by enclosing addresses in square brackets or by using an alternative syntax: [*bind_address*/]*port*. By default, the local port is bound in accordance with the **GatewayPorts** setting. However, an explicit *bind_address* may be used to bind the connection to a specific address. The *bind_address* of “localhost” indicates that the listening port be bound for local use only, while an empty address or ‘*’ indicates that the port should be available from all interfaces.

Currently the SOCKS4 and SOCKS5 protocols are supported, and ssh(1) will act as a SOCKS server. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the superuser can forward privileged ports.

EnableSSHKeysign

Setting this option to “yes” in the global client configuration file /etc/ssh/ssh_config enables the use of the helper program ssh-keysign(8) during **HostbasedAuthentication**. The argument must be “yes” or “no”. The default is “no”. This option should be placed in the non-hostspecific section. See ssh-keysign(8) for more information.

EscapeChar

Sets the escape character (default: ‘~’). The escape character can also be set on the command line. The argument should be a single character, ‘^’ followed by a letter, or “none” to disable the escape character entirely (making the connection transparent for binary data).

ExitOnForwardFailure

Specifies whether ssh(1) should terminate the connection if it cannot set up all requested dynamic, tunnel, local, and remote port forwardings. The argument must be “yes” or “no”. The default is “no”.

ForwardAgent

Specifies whether the connection to the authentication agent (if any) will be forwarded to the remote machine. The argument must be “yes” or “no”. The default is “no”.

Agent forwarding should be enabled with caution. Users with the ability to bypass file permissions on the remote host (for the agent’s Unix-domain socket) can access the local agent through the forwarded connection. An attacker cannot obtain key material from the agent, however they can perform operations on the keys that enable them to authenticate using the identities loaded into the agent.

ForwardX11
Specifies whether X11 connections will be automatically redirected over the secure channel and DISPLAY set. The argument must be “yes” or “no”. The default is “no”.

X11 forwarding should be enabled with caution. Users with the ability to bypass file permissions on the remote host (for the user’s X11 authorization database) can access the local X11 display through the forwarded connection. An attacker may then be able to perform activities such as keystroke monitoring if the **ForwardX11Trusted** option is also enabled.

ForwardX11Trusted
If this option is set to “yes”, remote X11 clients will have full access to the original X11 display.

If this option is set to “no”, remote X11 clients will be considered untrusted and prevented from stealing or tampering with data belonging to trusted X11 clients. Furthermore, the xauth(1) token used for the session will be set to expire after 20 minutes. Remote clients will be refused access after this time.

The default is “no”.

See the X11 SECURITY extension specification for full details on the restrictions imposed on untrusted clients.

GatewayPorts
Specifies whether remote hosts are allowed to connect to local forwarded ports. By default, ssh(1) binds local port forwardings to the loopback address. This prevents other remote hosts from connecting to forwarded ports. **GatewayPorts** can be used to specify that ssh should bind local port forwardings to the wildcard address, thus allowing remote hosts to connect to forwarded ports. The argument must be “yes” or “no”. The default is “no”.

GlobalKnownHostsFile
Specifies a file to use for the global host key database instead of /etc/ssh/ssh_known_hosts.

GSSAPIAuthentication
Specifies whether user authentication based on GSSAPI is allowed. The default is “no”. Note that this option applies to protocol version 2 only.

GSSAPIDelegateCredentials
Forward (delegate) credentials to the server. The default is “no”. Note that this option applies to protocol version 2 only.

HashKnownHosts
Indicates that ssh(1) should hash host names and addresses when they are added to ~/.ssh/known_hosts. These hashed names may be used normally by ssh(1) and sshd(8), but they do not reveal identifying information should the file’s contents be disclosed. The default is “no”. Note that existing names and addresses in known hosts files will not be converted automatically, but may be manually hashed using ssh-keygen(1).

HostbasedAuthentication
Specifies whether to try rhosts based authentication with public key authentication. The argument must be “yes” or “no”. The default is “no”. This option applies to protocol version 2 only and is similar to **RhostsRSAAuthentication**.

HostKeyAlgorithms
Specifies the protocol version 2 host key algorithms that the client wants to use in order of preference. The default for this option is: “ssh-rsa,ssh-dss”.

HostKeyAlias
Specifies an alias that should be used instead of the real host name when looking up or saving the host key in the host key database files. This option is useful for tunneling SSH connections or for multiple servers running on a single host.

HostName
Specifies the real host name to log into. This can be used to specify nicknames or abbreviations for hosts. The default is the name given on the command line. Numeric IP addresses are also permitted (both on the command line and in **HostName** specifications).

IdentitiesOnly
Specifies that ssh(1) should only use the authentication identity files configured in the **ssh_config** files, even if ssh-agent(1) offers more identities. The argument to this keyword must be "yes" or "no". This option is intended for situations where ssh-agent offers many different identities. The default is "no".

IdentityFile
Specifies a file from which the user's RSA or DSA authentication identity is read. The default is ~/.ssh/identity for protocol version 1, and ~/.ssh/id_rsa and ~/.ssh/id_dsa for protocol version 2. Additionally, any identities represented by the authentication agent will be used for authentication.

The file name may use the tilde syntax to refer to a user's home directory or one of the following escape characters: '%d' (local user's home directory), '%u' (local user name), '%l' (local host name), '%h' (remote host name) or '%r' (remote user name).

It is possible to have multiple identity files specified in configuration files; all these identities will be tried in sequence.

KbdInteractiveAuthentication
Specifies whether to use keyboard-interactive authentication. The argument to this keyword must be "yes" or "no". The default is "yes".

KbdInteractiveDevices
Specifies the list of methods to use in keyboard-interactive authentication. Multiple method names must be comma-separated. The default is to use the server specified list. The methods available vary depending on what the server supports. For an OpenSSH server, it may be zero or more of: "bsdauth", "pam", and "skey".

LocalCommand
Specifies a command to execute on the local machine after successfully connecting to the server. The command string extends to the end of the line, and is executed with the user's shell. This directive is ignored unless **PermitLocalCommand** has been enabled.

LocalForward
Specifies that a TCP port on the local machine be forwarded over the secure channel to the specified host and port from the remote machine. The first argument must be [*bind_address*:]*port* and the second argument must be *host*:*hostport*. IPv6 addresses can be specified by enclosing addresses in square brackets or by using an alternative syntax: [*bind_address*/]*port* and *host*/*hostport*. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the superuser can forward privileged ports. By default, the local port is bound in accordance with the **GatewayPorts** setting. However, an explicit *bind_address* may be used to bind the connection to a specific address. The *bind_address* of "localhost" indicates that the listening port be bound for local use only, while an empty address or '*' indicates that the port should be available from all interfaces.

LogLevel
Gives the verbosity level that is used when logging messages from ssh(1). The possible values are: QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG, DEBUG1, DEBUG2, and DEBUG3. The default is INFO. DEBUG and DEBUG1 are equivalent. DEBUG2 and DEBUG3 each specify higher levels of verbose output.

MACs Specifies the MAC (message authentication code) algorithms in order of preference. The MAC algorithm is used in protocol version 2 for data integrity protection. Multiple algorithms must be comma-separated. The default is:

```
hmac-md5,hmac-sha1,umac-64@openssh.com,
hmac-ripemd160,hmac-sha1-96,hmac-md5-96
```

NoHostAuthenticationForLocalhost
This option can be used if the home directory is shared across machines. In this case localhost will refer to a different machine on each of the machines and the user will get many warnings about changed host keys. However, this option disables host authentication for localhost. The argument to this keyword must be “yes” or “no”. The default is to check the host key for localhost.

NumberOfPasswordPrompts
Specifies the number of password prompts before giving up. The argument to this keyword must be an integer. The default is 3.

PasswordAuthentication
Specifies whether to use password authentication. The argument to this keyword must be “yes” or “no”. The default is “yes”.

PermitLocalCommand
Allow local command execution via the **LocalCommand** option or using the **!***command* escape sequence in ssh(1). The argument must be “yes” or “no”. The default is “no”.

Port Specifies the port number to connect on the remote host. The default is 22.

PreferredAuthentications
Specifies the order in which the client should try protocol 2 authentication methods. This allows a client to prefer one method (e.g. **keyboard-interactive**) over another method (e.g. **password**) The default for this option is: “gssapi-with-mic, hostbased, publickey, keyboard-interactive, password”.

Protocol
Specifies the protocol versions ssh(1) should support in order of preference. The possible values are ‘1’ and ‘2’. Multiple versions must be comma-separated. The default is “2,1”. This means that ssh tries version 2 and falls back to version 1 if version 2 is not available.

ProxyCommand
Specifies the command to use to connect to the server. The command string extends to the end of the line, and is executed with the user’s shell. In the command string, ‘%h’ will be substituted by the host name to connect and ‘%p’ by the port. The command can be basically anything, and should read from its standard input and write to its standard output. It should eventually connect an sshd(8) server running on some machine, or execute **sshd -i** somewhere. Host key management will be done using the HostName of the host being connected (defaulting to the name typed by the user). Setting the command to “none” disables this option entirely. Note that **CheckHostIP** is not available for connects with a proxy command.

This directive is useful in conjunction with nc(1) and its proxy support. For example, the following directive would connect via an HTTP proxy at 192.0.2.0:

```
ProxyCommand /usr/bin/nc -X connect -x 192.0.2.0:8080 %h %p
```

PubkeyAuthentication
Specifies whether to try public key authentication. The argument to this keyword must be “yes” or “no”. The default is “yes”. This option applies to protocol version 2 only.

RekeyLimit
Specifies the maximum amount of data that may be transmitted before the session key is renegotiated. The argument is the number of bytes, with an optional suffix of ‘K’, ‘M’, or ‘G’ to indicate Kilobytes, Megabytes, or Gigabytes, respectively. The default is between ‘1G’ and ‘4G’, depending on the cipher. This option applies to protocol version 2 only.

RemoteForward
Specifies that a TCP port on the remote machine be forwarded over the secure channel to the specified host and port from the local machine. The first argument must be [*bind_address*:]*port* and the second argument must be *host*:*hostport*. IPv6 addresses can be specified by enclosing addresses in square brackets or by using an alternative syntax: [*bind_address*/]*port* and *host*/*hostport*. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the superuser can forward privileged ports.

If the *bind_address* is not specified, the default is to only bind to loopback addresses. If the *bind_address* is ‘*’ or an empty string, then the forwarding is requested to listen on all interfaces. Specifying a remote *bind_address* will only succeed if the server’s **GatewayPorts** option is enabled (see sshd_config(5)).

RhostsRSAAuthentication
Specifies whether to try rhosts based authentication with RSA host authentication. The argument must be “yes” or “no”. The default is “no”. This option applies to protocol version 1 only and requires ssh(1) to be setuid root.

RSAAuthentication
Specifies whether to try RSA authentication. The argument to this keyword must be “yes” or “no”. RSA authentication will only be attempted if the identity file exists, or an authentication agent is running. The default is “yes”. Note that this option applies to protocol version 1 only.

SendEnv
Specifies what variables from the local environ(7) should be sent to the server. Note that environment passing is only supported for protocol 2. The server must also support it, and the server must be configured to accept these environment variables. Refer to **AcceptEnv** in sshd_config(5) for how to configure the server. Variables are specified by name, which may contain wildcard characters. Multiple environment variables may be separated by whitespace or spread across multiple **SendEnv** directives. The default is not to send any environment variables.

See **PATTERNS** for more information on patterns.

ServerAliveCountMax
Sets the number of server alive messages (see below) which may be sent without ssh(1) receiving any messages back from the server. If this threshold is reached while server alive messages are being sent, ssh will disconnect from the server, terminating the session. It is important to note that the use of server alive messages is very different from **TCPKeepAlive** (below). The server alive messages are sent through the encrypted channel and therefore will not be spoofable. The TCP keepalive option enabled by **TCPKeepAlive** is spoofable. The server alive mechanism is valuable when the client or server depend on knowing when a connection has become inactive.

The default value is 3. If, for example, **ServerAliveInterval** (see below) is set to 15 and **ServerAliveCountMax** is left at the default, if the server becomes unresponsive, ssh will disconnect after approximately 45 seconds. This option applies to protocol version 2 only.

ServerAliveInterval
Sets a timeout interval in seconds after which if no data has been received from the server, ssh(1) will send a message through the encrypted channel to request a response from the server. The default is 0, indicating that these messages will not be sent to the server. This option applies to protocol version 2 only.

SmartcardDevice
Specifies which smartcard device to use. The argument to this keyword is the device ssh(1) should use to communicate with a smartcard used for storing the user's private RSA key. By default, no device is specified and smartcard support is not activated.

StrictHostKeyChecking
If this flag is set to "yes", ssh(1) will never automatically add host keys to the ~/.ssh/known_hosts file, and refuses to connect to hosts whose host key has changed. This provides maximum protection against trojan horse attacks, though it can be annoying when the /etc/ssh/ssh_known_hosts file is poorly maintained or when connections to new hosts are frequently made. This option forces the user to manually add all new hosts. If this flag is set to "no", ssh will automatically add new host keys to the user known hosts files. If this flag is set to "ask", new host keys will be added to the user known host files only after the user has confirmed that is what they really want to do, and ssh will refuse to connect to hosts whose host key has changed. The host keys of known hosts will be verified automatically in all cases. The argument must be "yes", "no", or "ask". The default is "ask".

TCPKeepAlive
Specifies whether the system should send TCP keepalive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed. However, this means that connections will die if the route is down temporarily, and some people find it annoying.

The default is "yes" (to send TCP keepalive messages), and the client will notice if the network goes down or the remote host dies. This is important in scripts, and many users want it too.

To disable TCP keepalive messages, the value should be set to "no".

Tunnel
Request tun(4) device forwarding between the client and the server. The argument must be "yes", "point-to-point" (layer 3), "ethernet" (layer 2), or "no". Specifying "yes" requests the default tunnel mode, which is "point-to-point". The default is "no".

TunnelDevice
Specifies the tun(4) devices to open on the client (*local_tun*) and the server (*remote_tun*).

The argument must be *local_tun*[:*remote_tun*]. The devices may be specified by numerical ID or the keyword "any", which uses the next available tunnel device. If *remote_tun* is not specified, it defaults to "any". The default is "any:any".

UsePrivilegedPort
Specifies whether to use a privileged port for outgoing connections. The argument must be "yes" or "no". The default is "no". If set to "yes", ssh(1) must be setuid root. Note that this option must be set to "yes" for **RhostsRSAAuthentication** with older servers.

User Specifies the user to log in as. This can be useful when a different user name is used on different machines. This saves the trouble of having to remember to give the user name on the command line.

UserKnownHostsFile
Specifies a file to use for the user host key database instead of `~/.ssh/known_hosts`.

VerifyHostKeyDNS
Specifies whether to verify the remote key using DNS and SSHFP resource records. If this option is set to "yes", the client will implicitly trust keys that match a secure fingerprint from DNS. Insecure fingerprints will be handled as if this option was set to "ask". If this option is set to "ask", information on fingerprint match will be displayed, but the user will still need to confirm new host keys according to the **StrictHostKeyChecking** option. The argument must be "yes", "no", or "ask". The default is "no". Note that this option applies to protocol version 2 only.

See also **VERIFYING HOST KEYS** in `ssh`(1).

XAuthLocation
Specifies the full pathname of the `xauth`(1) program. The default is `/usr/X11R6/bin/xauth`.

PATTERNS

A *pattern* consists of zero or more non-whitespace characters, '*' (a wildcard that matches zero or more characters), or '?' (a wildcard that matches exactly one character). For example, to specify a set of declarations for any host in the ".co.uk" set of domains, the following pattern could be used:

```
Host *.co.uk
```

The following pattern would match any host in the 192.168.0.[0-9] network range:

```
Host 192.168.0.?
```

A *pattern-list* is a comma-separated list of patterns. Patterns within pattern-lists may be negated by preceding them with an exclamation mark ('!'). For example, to allow a key to be used from anywhere within an organisation except from the "dialup" pool, the following entry (in authorized_keys) could be used:

```
from="!*.dialup.example.com,*.example.com"
```

FILES

`~/.ssh/config`
This is the per-user configuration file. The format of this file is described above. This file is used by the SSH client. Because of the potential for abuse, this file must have strict permissions: read/write for the user, and not accessible by others.

`/etc/ssh/ssh_config`
Systemwide configuration file. This file provides defaults for those values that are not specified in the user's configuration file, and for those users who do not have a configuration file. This file must be world-readable.

SEE ALSO

`ssh`(1)

AUTHORS

OpenSSH is a derivative of the original and free ssh 1.2.12 release by Tatu Ylonen. Aaron Campbell, Bob Beck, Markus Friedl, Niels Provos, Theo de Raadt and Dug Song removed many bugs, re-added newer features and created OpenSSH. Markus Friedl contributed the support for SSH protocol versions 1.5 and 2.0.

NAME

sshd_config — OpenSSH SSH daemon configuration file

SYNOPSIS

/etc/ssh/sshd_config

DESCRIPTION

sshd(8) reads configuration data from /etc/ssh/sshd_config (or the file specified with **-f** on the command line). The file contains keyword-argument pairs, one per line. Lines starting with '#' and empty lines are interpreted as comments. Arguments may optionally be enclosed in double quotes (") in order to represent arguments containing spaces.

The possible keywords and their meanings are as follows (note that keywords are case-insensitive and arguments are case-sensitive):

AcceptEnv

Specifies what environment variables sent by the client will be copied into the session's environ(7). See **SendEnv** in ssh_config(5) for how to configure the client. Note that environment passing is only supported for protocol 2. Variables are specified by name, which may contain the wildcard characters '*' and '?'. Multiple environment variables may be separated by whitespace or spread across multiple **AcceptEnv** directives. Be warned that some environment variables could be used to bypass restricted user environments. For this reason, care should be taken in the use of this directive. The default is not to accept any environment variables.

AddressFamily

Specifies which address family should be used by sshd(8). Valid arguments are "any", "inet" (use IPv4 only), or "inet6" (use IPv6 only). The default is "any".

AllowGroups

This keyword can be followed by a list of group name patterns, separated by spaces. If specified, login is allowed only for users whose primary group or supplementary group list matches one of the patterns. Only group names are valid; a numerical group ID is not recognized. By default, login is allowed for all groups. The allow/deny directives are processed in the following order: **DenyUsers**, **AllowUsers**, **DenyGroups**, and finally **AllowGroups**.

See **PATTERNS** in ssh_config(5) for more information on patterns.

AllowTcpForwarding

Specifies whether TCP forwarding is permitted. The default is "yes". Note that disabling TCP forwarding does not improve security unless users are also denied shell access, as they can always install their own forwarders.

AllowUsers

This keyword can be followed by a list of user name patterns, separated by spaces. If specified, login is allowed only for user names that match one of the patterns. Only user names are valid; a numerical user ID is not recognized. By default, login is allowed for all users. If the pattern takes the form USER@HOST then USER and HOST are separately checked, restricting logins to particular users from particular hosts. The allow/deny directives are processed in the following order: **DenyUsers**, **AllowUsers**, **DenyGroups**, and finally **AllowGroups**.

See **PATTERNS** in ssh_config(5) for more information on patterns.

AuthorizedKeysFile

Specifies the file that contains the public keys that can be used for user authentication. **AuthorizedKeysFile** may contain tokens of the form %T which are substituted during connection setup. The following tokens are defined: %% is replaced by a literal '%', %h is replaced by the

home directory of the user being authenticated, and %u is replaced by the username of that user. After expansion, **AuthorizedKeysFile** is taken to be an absolute path or one relative to the user's home directory. The default is ".ssh/authorized_keys".

Banner
The contents of the specified file are sent to the remote user before authentication is allowed. If the argument is "none" then no banner is displayed. This option is only available for protocol version 2. By default, no banner is displayed.

ChallengeResponseAuthentication
Specifies whether challenge-response authentication is allowed. All authentication styles from `login.conf`(5) are supported. The default is "yes".

ChrootDirectory
Specifies a path to `chroot`(2) to after authentication. This path, and all its components, must be root-owned directories that are not writable by any other user or group.

The path may contain the following tokens that are expanded at runtime once the connecting user has been authenticated: %% is replaced by a literal '%', %h is replaced by the home directory of the user being authenticated, and %u is replaced by the username of that user.

The **ChrootDirectory** must contain the necessary files and directories to support the users' session. For an interactive session this requires at least a shell, typically `sh`(1), and basic `/dev` nodes such as `null`(4), `zero`(4), `stdin`(4), `stdout`(4), `stderr`(4), `arandom`(4) and `tty`(4) devices. For file transfer sessions using "sftp", no additional configuration of the environment is necessary if the in-process sftp server is used (see **Subsystem** for details).

The default is not to `chroot`(2).

Ciphers
Specifies the ciphers allowed for protocol version 2. Multiple ciphers must be comma-separated. The supported ciphers are "3des-cbc", "aes128-cbc", "aes192-cbc", "aes256-cbc", "aes128-ctr", "aes192-ctr", "aes256-ctr", "arcfour128", "arcfour256", "arcfour", "blowfish-cbc", and "cast128-cbc". The default is:

```
aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour128,
arcfour256,arcfour,aes192-cbc,aes256-cbc,aes128-ctr,
aes192-ctr,aes256-ctr
```

ClientAliveCountMax
Sets the number of client alive messages (see below) which may be sent without `sshd`(8) receiving any messages back from the client. If this threshold is reached while client alive messages are being sent, sshd will disconnect the client, terminating the session. It is important to note that the use of client alive messages is very different from **TCPKeepAlive** (below). The client alive messages are sent through the encrypted channel and therefore will not be spoofable. The TCP keepalive option enabled by **TCPKeepAlive** is spoofable. The client alive mechanism is valuable when the client or server depend on knowing when a connection has become inactive.

The default value is 3. If **ClientAliveInterval** (see below) is set to 15, and **ClientAliveCountMax** is left at the default, unresponsive SSH clients will be disconnected after approximately 45 seconds. This option applies to protocol version 2 only.

ClientAliveInterval
Sets a timeout interval in seconds after which if no data has been received from the client, `sshd`(8) will send a message through the encrypted channel to request a response from the client. The default is 0, indicating that these messages will not be sent to the client. This option applies to protocol version 2 only.

Compression
Specifies whether compression is allowed, or delayed until the user has authenticated successfully. The argument must be “yes”, “delayed”, or “no”. The default is “delayed”.

DenyGroups
This keyword can be followed by a list of group name patterns, separated by spaces. Login is disallowed for users whose primary group or supplementary group list matches one of the patterns. Only group names are valid; a numerical group ID is not recognized. By default, login is allowed for all groups. The allow/deny directives are processed in the following order: **DenyUsers**, **AllowUsers**, **DenyGroups**, and finally **AllowGroups**.

See **PATTERNS** in ssh_config(5) for more information on patterns.

DenyUsers
This keyword can be followed by a list of user name patterns, separated by spaces. Login is disallowed for user names that match one of the patterns. Only user names are valid; a numerical user ID is not recognized. By default, login is allowed for all users. If the pattern takes the form USER@HOST then USER and HOST are separately checked, restricting logins to particular users from particular hosts. The allow/deny directives are processed in the following order: **DenyUsers**, **AllowUsers**, **DenyGroups**, and finally **AllowGroups**.

See **PATTERNS** in ssh_config(5) for more information on patterns.

ForceCommand
Forces the execution of the command specified by **ForceCommand**, ignoring any command supplied by the client and ~/.ssh/rc if present. The command is invoked by using the user’s login shell with the -c option. This applies to shell, command, or subsystem execution. It is most useful inside a **Match** block. The command originally supplied by the client is available in the SSH_ORIGINAL_COMMAND environment variable. Specifying a command of “internal-sftp” will force the use of an in-process sftp server that requires no support files when used with **ChrootDirectory**.

GatewayPorts
Specifies whether remote hosts are allowed to connect to ports forwarded for the client. By default, sshd(8) binds remote port forwardings to the loopback address. This prevents other remote hosts from connecting to forwarded ports. **GatewayPorts** can be used to specify that sshd should allow remote port forwardings to bind to non-loopback addresses, thus allowing other hosts to connect. The argument may be “no” to force remote port forwardings to be available to the local host only, “yes” to force remote port forwardings to bind to the wildcard address, or “clientspecified” to allow the client to select the address to which the forwarding is bound. The default is “no”.

GSSAPIAuthentication
Specifies whether user authentication based on GSSAPI is allowed. The default is “no”. Note that this option applies to protocol version 2 only.

GSSAPICleanupCredentials
Specifies whether to automatically destroy the user’s credentials cache on logout. The default is “yes”. Note that this option applies to protocol version 2 only.

HostbasedAuthentication
Specifies whether rhosts or /etc/hosts.equiv authentication together with successful public key client host authentication is allowed (host-based authentication). This option is similar to **RhostsRSAAuthentication** and applies to protocol version 2 only. The default is “no”.

HostbasedUsesNameFromPacketOnly
Specifies whether or not the server will attempt to perform a reverse name lookup when matching the name in the ~/.shosts, ~/.rhosts, and /etc/hosts.equiv files during

HostbasedAuthentication. A setting of “yes” means that sshd(8) uses the name supplied by the client rather than attempting to resolve the name from the TCP connection itself. The default is “no”.

HostKey
Specifies a file containing a private host key used by SSH. The default is /etc/ssh/ssh_host_key for protocol version 1, and /etc/ssh/ssh_host_rsa_key and /etc/ssh/ssh_host_dsa_key for protocol version 2. Note that sshd(8) will refuse to use a file if it is group/world-accessible. It is possible to have multiple host key files. “rsa1” keys are used for version 1 and “dsa” or “rsa” are used for version 2 of the SSH protocol.

IgnoreRhosts
Specifies that .rhosts and .shosts files will not be used in **RhostsRSAAuthentication** or **HostbasedAuthentication**.

/etc/hosts.equiv and /etc/shosts.equiv are still used. The default is “yes”.

IgnoreUserKnownHosts
Specifies whether sshd(8) should ignore the user’s ~/.ssh/known_hosts during **RhostsRSAAuthentication** or **HostbasedAuthentication**. The default is “no”.

KerberosAuthentication
Specifies whether the password provided by the user for **PasswordAuthentication** will be validated through the Kerberos KDC. To use this option, the server needs a Kerberos servtab which allows the verification of the KDC’s identity. The default is “no”.

KerberosGetAFSToken
If AFS is active and the user has a Kerberos 5 TGT, attempt to acquire an AFS token before accessing the user’s home directory. The default is “no”.

KerberosOrLocalPasswd
If password authentication through Kerberos fails then the password will be validated via any additional local mechanism such as /etc/passwd. The default is “yes”.

KerberosTicketCleanup
Specifies whether to automatically destroy the user’s ticket cache file on logout. The default is “yes”.

KeyRegenerationInterval
In protocol version 1, the ephemeral server key is automatically regenerated after this many seconds (if it has been used). The purpose of regeneration is to prevent decrypting captured sessions by later breaking into the machine and stealing the keys. The key is never stored anywhere. If the value is 0, the key is never regenerated. The default is 3600 (seconds).

ListenAddress
Specifies the local addresses sshd(8) should listen on. The following forms may be used:

ListenAddress *host*|*IPv4_addr*|*IPv6_addr*
ListenAddress *host*|*IPv4_addr*:*port*
ListenAddress [*host*|*IPv6_addr*]:*port*

If *port* is not specified, sshd will listen on the address and all prior **Port** options specified. The default is to listen on all local addresses. Multiple **ListenAddress** options are permitted. Additionally, any **Port** options must precede this option for non-port qualified addresses.

LoginGraceTime
The server disconnects after this time if the user has not successfully logged in. If the value is 0, there is no time limit. The default is 120 seconds.

LogLevel
Gives the verbosity level that is used when logging messages from sshd(8). The possible values are: QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG, DEBUG1, DEBUG2, and DEBUG3. The default is INFO. DEBUG and DEBUG1 are equivalent. DEBUG2 and DEBUG3 each specify higher levels of debugging output. Logging with a DEBUG level violates the privacy of users and is not recommended.

MACs Specifies the available MAC (message authentication code) algorithms. The MAC algorithm is used in protocol version 2 for data integrity protection. Multiple algorithms must be comma-separated. The default is:

```
hmac-md5,hmac-sha1,umac-64@openssh.com,
hmac-ripemd160,hmac-sha1-96,hmac-md5-96
```

Match Introduces a conditional block. If all of the criteria on the **Match** line are satisfied, the keywords on the following lines override those set in the global section of the config file, until either another **Match** line or the end of the file. The arguments to **Match** are one or more criteria-pattern pairs. The available criteria are **User**, **Group**, **Host**, and **Address**. Only a subset of keywords may be used on the lines following a **Match** keyword. Available keywords are **AllowTcpForwarding**, **Banner**, **ForceCommand**, **GatewayPorts**, **GSSApiAuthentication**, **KbdInteractiveAuthentication**, **KerberosAuthentication**, **PasswordAuthentication**, **PermitOpen**, **PermitRootLogin**, **RhostsRSAAuthentication**, **RSAAuthentication**, **X11DisplayOffset**, **X11Forwarding**, and **X11UseLocalHost**.

MaxAuthTries
Specifies the maximum number of authentication attempts permitted per connection. Once the number of failures reaches half this value, additional failures are logged. The default is 6.

MaxStartups
Specifies the maximum number of concurrent unauthenticated connections to the SSH daemon. Additional connections will be dropped until authentication succeeds or the **LoginGraceTime** expires for a connection. The default is 10.

Alternatively, random early drop can be enabled by specifying the three colon separated values “start:rate:full” (e.g. "10:30:60"). sshd(8) will refuse connection attempts with a probability of “rate/100” (30%) if there are currently “start” (10) unauthenticated connections. The probability increases linearly and all connection attempts are refused if the number of unauthenticated connections reaches “full” (60).

PasswordAuthentication
Specifies whether password authentication is allowed. The default is “yes”.

PermitEmptyPasswords
When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings. The default is “no”.

PermitOpen
Specifies the destinations to which TCP port forwarding is permitted. The forwarding specification must be one of the following forms:

PermitOpen *host*:*port*
PermitOpen *IPv4_addr*:*port*
PermitOpen *[IPv6_addr]*:*port*

Multiple forwards may be specified by separating them with whitespace. An argument of “any” can be used to remove all restrictions and permit any forwarding requests. By default all port forward-

ing requests are permitted.

PermitRootLogin
Specifies whether root can log in using ssh(1). The argument must be “yes”, “without-password”, “forced-commands-only”, or “no”. The default is “no”.

If this option is set to “without-password”, password authentication is disabled for root.

If this option is set to “forced-commands-only”, root login with public key authentication will be allowed, but only if the *command* option has been specified (which may be useful for taking remote backups even if root login is normally not allowed). All other authentication methods are disabled for root.

If this option is set to “no”, root is not allowed to log in.

PermitTunnel
Specifies whether tun(4) device forwarding is allowed. The argument must be “yes”, “point-to-point” (layer 3), “ethernet” (layer 2), or “no”. Specifying “yes” permits both “point-to-point” and “ethernet”. The default is “no”.

PermitUserEnvironment
Specifies whether ~/.ssh/environment and **environment=** options in ~/.ssh/authorized_keys are processed by sshd(8). The default is “no”. Enabling environment processing may enable users to bypass access restrictions in some configurations using mechanisms such as LD_PRELOAD.

PidFile
Specifies the file that contains the process ID of the SSH daemon. The default is /var/run/sshd.pid.

Port Specifies the port number that sshd(8) listens on. The default is 22. Multiple options of this type are permitted. See also **ListenAddress**.

PrintLastLog
Specifies whether sshd(8) should print the date and time of the last user login when a user logs in interactively. The default is “yes”.

PrintMotd
Specifies whether sshd(8) should print /etc/motd when a user logs in interactively. (On some systems it is also printed by the shell, /etc/profile, or equivalent.) The default is “yes”.

Protocol
Specifies the protocol versions sshd(8) supports. The possible values are ‘1’ and ‘2’. Multiple versions must be comma-separated. The default is “2,1”. Note that the order of the protocol list does not indicate preference, because the client selects among multiple protocol versions offered by the server. Specifying “2,1” is identical to “1,2”.

PubkeyAuthentication
Specifies whether public key authentication is allowed. The default is “yes”. Note that this option applies to protocol version 2 only.

RhostsRSAAuthentication
Specifies whether rhosts or /etc/hosts.equiv authentication together with successful RSA host authentication is allowed. The default is “no”. This option applies to protocol version 1 only.

RSAAuthentication
Specifies whether pure RSA authentication is allowed. The default is “yes”. This option applies to protocol version 1 only.

ServerKeyBits
Defines the number of bits in the ephemeral protocol version 1 server key. The minimum value is 512, and the default is 768.

StrictModes
Specifies whether sshd(8) should check file modes and ownership of the user's files and home directory before accepting login. This is normally desirable because novices sometimes accidentally leave their directory or files world-writable. The default is "yes".

Subsystem
Configures an external subsystem (e.g. file transfer daemon). Arguments should be a subsystem name and a command (with optional arguments) to execute upon subsystem request.

The command sftp-server(8) implements the "sftp" file transfer subsystem.

Alternately the name "internal-sftp" implements an in-process "sftp" server. This may simplify configurations using **ChrootDirectory** to force a different filesystem root on clients.

By default no subsystems are defined. Note that this option applies to protocol version 2 only.

SyslogFacility
Gives the facility code that is used when logging messages from sshd(8). The possible values are: DAEMON, USER, AUTH, LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7. The default is AUTH.

TCPKeepAlive
Specifies whether the system should send TCP keepalive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed. However, this means that connections will die if the route is down temporarily, and some people find it annoying. On the other hand, if TCP keepalives are not sent, sessions may hang indefinitely on the server, leaving "ghost" users and consuming server resources.

The default is "yes" (to send TCP keepalive messages), and the server will notice if the network goes down or the client host crashes. This avoids infinitely hanging sessions.

To disable TCP keepalive messages, the value should be set to "no".

UseDNS
Specifies whether sshd(8) should look up the remote host name and check that the resolved host name for the remote IP address maps back to the very same IP address. The default is "yes".

UseLogin
Specifies whether login(1) is used for interactive login sessions. The default is "no". Note that login(1) is never used for remote command execution. Note also, that if this is enabled, **X11Forwarding** will be disabled because login(1) does not know how to handle xauth(1) cookies. If **UsePrivilegeSeparation** is specified, it will be disabled after authentication.

UsePrivilegeSeparation
Specifies whether sshd(8) separates privileges by creating an unprivileged child process to deal with incoming network traffic. After successful authentication, another process will be created that has the privilege of the authenticated user. The goal of privilege separation is to prevent privilege escalation by containing any corruption within the unprivileged processes. The default is "yes".

X11DisplayOffset
Specifies the first display number available for sshd(8)'s X11 forwarding. This prevents sshd from interfering with real X11 servers. The default is 10.

X11Forwarding
Specifies whether X11 forwarding is permitted. The argument must be “yes” or “no”. The default is “no”.

When X11 forwarding is enabled, there may be additional exposure to the server and to client displays if the sshd(8) proxy display is configured to listen on the wildcard address (see **X11UseLocalhost** below), though this is not the default. Additionally, the authentication spoofing and authentication data verification and substitution occur on the client side. The security risk of using X11 forwarding is that the client’s X11 display server may be exposed to attack when the SSH client requests forwarding (see the warnings for **ForwardX11** in ssh_config(5)). A system administrator may have a stance in which they want to protect clients that may expose themselves to attack by unwittingly requesting X11 forwarding, which can warrant a “no” setting.

Note that disabling X11 forwarding does not prevent users from forwarding X11 traffic, as users can always install their own forwarders. X11 forwarding is automatically disabled if **UseLogin** is enabled.

X11UseLocalhost
Specifies whether sshd(8) should bind the X11 forwarding server to the loopback address or to the wildcard address. By default, sshd binds the forwarding server to the loopback address and sets the hostname part of the DISPLAY environment variable to “localhost”. This prevents remote hosts from connecting to the proxy display. However, some older X11 clients may not function with this configuration. **X11UseLocalhost** may be set to “no” to specify that the forwarding server should be bound to the wildcard address. The argument must be “yes” or “no”. The default is “yes”.

XAuthLocation
Specifies the full pathname of the xauth(1) program. The default is /usr/X11R6/bin/xauth.

TIME FORMATS

sshd(8) command-line arguments and configuration file options that specify time may be expressed using a sequence of the form: *time*[*qualifier*], where *time* is a positive integer value and *qualifier* is one of the following:

⟨**none**⟩ seconds
s | **S** seconds
m | **M** minutes
h | **H** hours
d | **D** days
w | **W** weeks

Each member of the sequence is added together to calculate the total time value.

Time format examples:

600 600 seconds (10 minutes)
10m 10 minutes
1h30m 1 hour 30 minutes (90 minutes)

FILES

/etc/ssh/sshd_config
Contains configuration data for sshd(8). This file should be writable by root only, but it is recommended (though not necessary) that it be world-readable.

SEE ALSO

sshd(8)

AUTHORS

OpenSSH is a derivative of the original and free ssh 1.2.12 release by Tatu Ylonen. Aaron Campbell, Bob Beck, Markus Friedl, Niels Provos, Theo de Raadt and Dug Song removed many bugs, re-added newer features and created OpenSSH. Markus Friedl contributed the support for SSH protocol versions 1.5 and 2.0. Niels Provos and Markus Friedl contributed support for privilege separation.

NAME

stab — symbol table types

SYNOPSIS

```
#include <stab.h>
```

DESCRIPTION

The file ⟨stab.h⟩ defines some of the symbol table *n_type* field values for a.out files. These are the types for permanent symbols (i.e. not local labels, etc.) used by the old debugger *sdb* and the Berkeley Pascal compiler **pc**. Symbol table entries can be produced by the `.stabs` assembler directive. This allows one to specify a double-quote delimited name, a symbol type, one char and one short of information about the symbol, and an unsigned long (usually an address). To avoid having to produce an explicit label for the address field, the `.stabd` directive can be used to implicitly address the current location. If no name is needed, symbol table entries can be generated using the `.stabn` directive. The loader promises to preserve the order of symbol table entries produced by `.stab` directives. As described in a.out(5), an element of the symbol table consists of the following structure:

```
/*
* Format of a symbol table entry.
*/

struct nlist {
        union {
                char    *n_name;        /* for use when in-core */
                long    n_strx;         /* index into file string table */
        } n_un;
        unsigned char  n_type;          /* type flag */
        char           n_other;         /* unused */
        short          n_desc;          /* see struct desc, below */
        unsigned       n_value;         /* address or offset or line */
};
```

The low bits of the *n_type* field are used to place a symbol into at most one segment, according to the following masks, defined in ⟨a.out.h⟩. A symbol can be in none of these segments by having none of these segment bits set.

```
/*
* Simple values for n_type.
*/

#define N_UNDF  0x0     /* undefined */
#define N_ABS   0x2     /* absolute */
#define N_TEXT  0x4     /* text */
#define N_DATA  0x6     /* data */
#define N_BSS   0x8     /* bss */

#define N_EXT   01      /* external bit, or'ed in */
```

The *n_value* field of a symbol is relocated by the linker, ld(1) as an address within the appropriate segment. *n_value* fields of symbols not in any segment are unchanged by the linker. In addition, the linker will discard certain symbols, according to rules of its own, unless the *n_type* field has one of the following bits set:

```
/*
* Other permanent symbol table entries have some of the N_STAB bits set.
* These are given in <stab.h>
*/

#define N_STAB  0xe0    /* if any of these bits set, don't discard */
```

This allows up to 112 (7 * 16) symbol types, split between the various segments. Some of these have already been claimed. The old symbolic debugger, *sdb*, uses the following n_type values:

```
#define N_GSYM  0x20    /* global symbol: name,,0,type,0 */
#define N_FNAME 0x22    /* procedure name (f77 kludge): name,,0 */
#define N_FUN   0x24    /* procedure: name,,0,linenumber,address */
#define N_STSYM 0x26    /* static symbol: name,,0,type,address */
#define N_LCSYM 0x28    /* .lcomm symbol: name,,0,type,address */
#define N_RSYM  0x40    /* register sym: name,,0,type,register */
#define N_SLINE 0x44    /* src line: 0,,0,linenumber,address */
#define N_SSYM  0x60    /* structure elt: name,,0,type,struct_offset */
#define N_SO    0x64    /* source file name: name,,0,0,address */
#define N_LSYM  0x80    /* local sym: name,,0,type,offset */
#define N_SOL   0x84    /* #included file name: name,,0,0,address */
#define N_PSYM  0xa0    /* parameter: name,,0,type,offset */
#define N_ENTRY 0xa4    /* alternative entry: name,linenumber,address */
#define N_LBRAC 0xc0    /* left bracket: 0,,0,nesting level,address */
#define N_RBRAC 0xe0    /* right bracket: 0,,0,nesting level,address */
#define N_BCOMM 0xe2    /* begin common: name,, */
#define N_ECOMM 0xe4    /* end common: name,, */
#define N_ECOML 0xe8    /* end common (local name): ,,address */
#define N_LENG  0xfe    /* second stab entry with length information */
```

where the comments give *sdb* conventional use for `.stab` *s* and the *n_name*, *n_other*, *n_desc*, and *n_value* fields of the given *n_type*. *Sdb* uses the *n_desc* field to hold a type specifier in the form used by the Portable C Compiler, `cc`(1); see the header file `pcc.h` for details on the format of these type values.

The Berkeley Pascal compiler, **pc**, uses the following *n_type* value:

```
#define N_PC    0x30    /* global pascal symbol: name,,0,subtype,line */
```

and uses the following subtypes to do type checking across separately compiled files:

1	source file name
2	included file name
3	global label
4	global constant
5	global type
6	global variable
7	global function
8	global procedure
9	external function
10	external procedure
11	library variable
12	library routine

SEE ALSO

as(1), gdb(1), ld(1), a.out(5)

HISTORY

The **stab** file appeared in 4.0 BSD.

BUGS

More basic types are needed.

NAME

statvfs — file system statistics

SYNOPSIS

```
#include <sys/types.h>
#include <sys/statvfs.h>
```

DESCRIPTION

The ⟨sys/statvfs.h⟩ header defines the structures and functions that return information about a mounted file system. The **statvfs** structure is defined as follows:

```
typedef struct { int32_t val[2]; } fsid_t;   /* file system id type */

#define VFS_NAMELEN  32       /* length of fs type name, including nul */
#define VFS_MNAMELEN 1024     /* length of buffer for returned name */

struct statvfs {
        unsigned long  f_flag;   /* copy of mount exported flags */
        unsigned long  f_bsize;  /* system block size */
        unsigned long  f_frsize; /* system fragment size */
        unsigned long  f_iosize; /* optimal file system block size */

        fsblkcnt_t     f_blocks; /* number of blocks in file system */
        fsblkcnt_t     f_bfree;  /* free blocks avail in file system */
        fsblkcnt_t     f_bavail; /* free blocks avail to non-root */
        fsblkcnt_t     f_bresvd; /* blocks reserved for root */

        fsfilcnt_t     f_files;  /* total file nodes in file system */
        fsfilcnt_t     f_ffree;  /* free file nodes in file system */
        fsfilcnt_t     f_favail; /* free file nodes avail to non-root */
        fsfilcnt_t     f_fresvd; /* file nodes reserved for root */

        uint64_t  f_syncreads;   /* count of sync reads since mount */
        uint64_t  f_syncwrites;         /* count of sync writes since mount */

        uint64_t  f_asyncreads;         /* count of async reads since mount */
        uint64_t  f_asyncwrites;  /* count of async writes since mount */

        unsigned long  f_fsid;   /* POSIX compliant file system id */
        fsid_t          f_fsidx;  /* NetBSD compatible file system id */

        unsigned long  f_namemax;/* maximum filename length */
        uid_t          f_owner; /* user that mounted the file system */

        uint32_t       f_spare[4]; /* spare space */

        char    f_fstypename[VFS_NAMELEN]; /* fs type name */
        char    f_mntonname[VFS_MNAMELEN]; /* directory on which mounted */
        char    f_mntfromname[VFS_MNAMELEN]; /* mounted file system */
};
```

The *f_flag* argument can have the following bits set:

ST_MAGICLINKS	Expand special strings (beginning with "@") when traversing symbolic links. See symlink(7) for a list of supported strings.
ST_RDONLY	The filesystem is mounted read-only; Even the super-user may not write on it.
ST_NOEXEC	Files may not be executed from the filesystem.
ST_NOSUID	Setuid and setgid bits on files are not honored when they are executed.
ST_NODEV	Special files in the filesystem may not be opened.
ST_UNION	Union with underlying filesystem instead of obscuring it.
ST_SYNCHRONOUS	All I/O to the filesystem is done synchronously.
ST_ASYNC	No filesystem I/O is done synchronously.
ST_NOCOREDUMP	Don't write core dumps to this file system.
ST_NOATIME	Never update access times.
ST_SYMPERM	Recognize symbolic link permission.
ST_NODEVMTIME	Never update modification times for device files.
ST_SOFTDEP	Use soft dependencies.
ST_LOCAL	The filesystem resides locally.
ST_QUOTA	The filesystem has quotas enabled on it.
ST_ROOTFS	Identifies the root filesystem.
ST_EXRDONLY	The filesystem is exported read-only.
ST_EXPORTED	The filesystem is exported for both reading and writing.
ST_DEFEXPORTED	The filesystem is exported for both reading and writing to any Internet host.
ST_EXPORTANON	The filesystem maps all remote accesses to the anonymous user.
ST_EXKERB	The filesystem is exported with Kerberos uid mapping.
ST_EXNORESPORT	Don't enforce reserved ports (NFS).
ST_EXPUBLIC	Public export (WebNFS).

Fields that are undefined for a particular file system are set to −1.

NOTES

f_flag The *f_flag* field is the same as the *f_flags* field in the 4.3BSD statfs(2) system call.

f_fsid Is defined to be *unsigned long* by the X/Open standard. Unfortunately this is not enough space to store our *fsid_t*, so we define an additional *f_fsidx* field.

f_bavail Could historically be negative (in the statfs(2) system call) when the used space has exceeded the non-superuser free space. In order to comply with the X/Open standard, we have to define *fsblkcnt_t* as an unsigned type, so in all cases where *f_bavail* would have been negative, we set it to 0. In addition we provide *f_bresvd* which contains the amount of reserved blocks for the superuser, so the old value of *f_bavail* can be easily computed as:

```
old_bavail = f_bfree - f_bresvd;
```

SEE ALSO

`statvfs`(2)

HISTORY

The ⟨`sys/statvfs.h`⟩ header first appeared in NetBSD 3.0.

NAME

sysctl.conf — sysctl configuration file

SYNOPSIS

sysctl.conf

DESCRIPTION

The **sysctl.conf** file defines the sysctl(7) kernel state tunables that can be set at boot time using sysctl(8) (using the **-f** switch) via the /etc/rc.d/sysctl startup script.

The state to be set is described using a "Management Information Base" ("MIB") style name. The MIB and value must be separated by '=' with no whitespace, for example:

name=value

Blank lines, lines with just *name*, and comments (beginning with '#') are ignored. Line continuations using backslash '\' are permitted. Only integral and string values can be set.

FILES

/etc/sysctl.conf The file **sysctl.conf** resides in /etc.

EXAMPLES

The following is an example /etc/sysctl.conf file:

```
# Change max open files
kern.maxfiles=1792

# Run Veriexec in IDS mode
kern.veriexec.strict=1

# Enable IP packet forwarding
net.inet.ip.forwarding=1
```

SEE ALSO

sysctl(3), rc.conf(5), sysctl(7), sysctl(8)

HISTORY

Support for **sysctl.conf** first appeared in NetBSD 1.5.

NAME

syslog.conf — syslogd(8) configuration file

DESCRIPTION

The **syslog.conf** file is the configuration file for the syslogd(8) program. It consists of blocks of lines separated by *program* and *hostname* specifications, with each line containing two fields: the *selector* field which specifies the types of messages and priorities to which the line applies, and an *action* field which specifies the action to be taken if a message syslogd(8) receives matches the selection criteria. The *selector* field is separated from the *action* field by one or more tab characters.

The *Selectors* function are encoded as a *facility*, a period ('.'), an optional set of comparison flags ([!][<=>]), and a *level*, with no intervening white-space. Both the *facility* and the *level* are case insensitive.

The *facility* describes the part of the system generating the message, and is one of the following keywords: auth, authpriv, cron, ftp, daemon, kern, lpr, mail, mark, news, syslog, user, uucp and local0 through local7. These keywords (with the exception of mark) correspond to the similar "LOG_" values specified to the openlog(3) and syslog(3) library routines.

The *comparison flags* may be used to specify exactly what levels are logged. If unspecified, the default comparison is '>=' (greater than or equal to), or, if the **-U** option is passed to syslogd(8), '=' (equal to). Comparison flags beginning with '!' will have their logical sense inverted. Thus, '!=info' means all levels except info and '!notice' has the same meaning as '<notice'.

The *level* describes the severity of the message, and is a keyword from the following ordered list (higher to lower): emerg, alert, crit, err, warning, notice, info and debug. These keywords correspond to the similar (LOG_) values specified to the syslog(3) library routine.

Each block of lines is separated from the previous block by a *program* or *hostname* specification. A block will only log messages corresponding to the most recent *program* and *hostname* specifications given. Consider the case of a block that selects pppd as the *program*, directly followed by a block that selects messages from the *hostname* dialhost. The second block will log only messages from the pppd(8) program from the host 'dialhost'.

A *program* specification of the form #!+prog1,prog2 or !+prog1,prog2 will cause subsequent blocks to be applied to messages logged by the specified programs. A *program* specification of the form #!-prog1,prog2 or !-prog1,prog2 will cause subsequent blocks to be applied to messages logged by programs other than the ones specified. A *program* specification of the form #!prog1,prog2 or !prog1,prog2 is equivalent to !+prog1,prog2. Program selectors may also match kernel-generated messages. For example, a program specification of !+subsys will match kernel-generated messages of the form subsys: here is a message. The special specification '!*' will cause subsequent blocks to apply to all programs.

A *hostname* specification of the form #+host1,host2 or +host1,host2 will cause subsequent blocks to be applied to messages received from the specified hosts. A *hostname* specification of the form #-host1,host2 or -host1,host2 will cause subsequent blocks to be applied to messages from hosts other than the ones specified. If the hostname is given as '@', the local hostname will be used. The special specification '+*' will cause subsequent blocks to apply to all hosts.

See syslog(3) for a further descriptions of both the *facility* and *level* keywords and their significance. It is preferred that selections be made based on *facility* rather than *program*, since the latter can vary in a networked environment. However, there are cases where a *facility* may be too broadly defined.

If a received message matches the specified *facility*, and the specified *level* comparison is true, and the first word in the message after the date matches the *program*, the action specified in the *action* field will be taken.

Multiple *selectors* may be specified for a single *action* by separating them with semicolon (';') characters. It is important to note, however, that each *selector* can modify the ones preceding it.

Multiple *facilities* may be specified for a single *level* by separating them with comma (',') characters.

An asterisk ('*') can be used to specify all *facilities* or all *levels*.

The special *facility* "mark" receives a message at priority "info" every 20 minutes (see `syslogd`(8)). This is not enabled by a *facility* field containing an asterisk.

The special *level* "none" disables a particular *facility*.

The *action* field of each line specifies the action to be taken when the *selector* field selects a message. There are five forms:

- A pathname (beginning with a leading slash). Selected messages are appended to the file.

  To ensure that kernel messages are written to disk promptly, `syslogd`(8) calls `fsync`(2) after writing messages from the kernel. Other messages are not synced explcitly. You may disable syncing of files specified to receive kernel messages by prefixing the pathname with a minus sign '-'. Note that use of this option may cause the loss of log information in the event of a system crash immediately following the write attempt. However, using this option may prove to be useful if your system's kernel is logging many messages.

- A hostname (preceded by an at ('@') sign). Selected messages are forwarded to the `syslogd`(8) program on the named host.

- A comma separated list of users. Selected messages are written to those users if they are logged in.

- An asterisk. Selected messages are written to all logged-in users.

- A vertical bar ('|') followed by a command to which to pipe the selected messages. The command string is passed to `/bin/sh` for evaluation, so the usual shell metacharacters or input/output redirection can occur. (Note that redirecting `stdio`(3) buffered output from the invoked command can cause additional delays, or even lost output data in case a logging subprocess exits with a signal.) The command itself runs with *stdout* and *stderr* redirected to `/dev/null`. Upon receipt of a `SIGHUP`, `syslogd`(8) will close the pipe to the process. If the process does not exit voluntarily, it will be sent a `SIGTERM` signal after a grace period of up to 60 seconds.

  The command will only be started once data arrives that should be piped to it. If the command exits, it will be restarted as necessary.

  If it is desired that the subprocess should receive exactly one line of input, this can be achieved by exiting after reading and processing the single line. A wrapper script can be used to achieve this effect, if necessary. Note that this method can be very resource-intensive if many log messages are being piped through the filter.

  Unless the command is a full pipeline, it may be useful to start the command with *exec* so that the invoking shell process does not wait for the command to complete. Note that the command is started with the UID of the `syslogd`(8) process, normally the superuser.

Blank lines and lines whose first non-blank character is a hash ('#') character are ignored.

FILES

`/etc/syslog.conf` The `syslogd`(8) configuration file.

EXAMPLES

A configuration file might appear as follows:

```
# Log all kernel messages, authentication messages of
# level notice or higher and anything of level err or
# higher to the console.
# Don't log private authentication messages!
*.err;kern.*;auth.notice;authpriv.none       /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none       /var/log/messages

# Log daemon messages at debug level only
daemon.=debug                        /var/log/daemon.debug

# The authpriv file has restricted access.
authpriv.*                           /var/log/secure

# Log all the mail messages in one place.
mail.*                               /var/log/maillog

# Everybody gets emergency messages, plus log them on another
# machine.
*.emerg                              *
*.emerg                              @arpa.berkeley.edu

# Root and Eric get alert and higher messages.
*.alert                              root,eric

# Save mail and news errors of level err and higher in a
# special file.
mail,news.err                        /var/log/spoolerr

# Pipe all authentication messages to a filter.
auth.*                               |exec /usr/local/sbin/authfilter

# Log kernel messages to a separate file without syncing each message.
kern.*                               -/var/log/kernlog

# Save ftpd transactions along with mail and news.
!ftpd
*.*                                  /var/log/spoolerr

# Send all error messages from a RAID array through a filter.
!raid0
kern.err                             |exec /usr/local/sbin/raidfilter

# Save pppd messages from dialhost to a separate file.
!pppd
+dialhost
*.*                                  /var/log/dialhost-pppd

# Save non-local log messages from all programs to a separate file.
!*
```

```
-@
*.*                                     /var/log/foreign
```

SEE ALSO

`syslog`(3), `syslogd`(8)

HISTORY

The **`syslog.conf`** file appeared in 4.3BSD, along with `syslogd`(8).

BUGS

The effects of multiple selectors are sometimes not intuitive. For example “mail.crit;*.err” will select “mail” facility messages at the level of “err” or higher, not at the level of “crit” or higher.

NAME

tar — format of tape archive files

DESCRIPTION

The **tar** archive format collects any number of files, directories, and other file system objects (symbolic links, device nodes, etc.) into a single stream of bytes. The format was originally designed to be used with tape drives that operate with fixed-size blocks, but is widely used as a general packaging mechanism.

General Format

A **tar** archive consists of a series of 512-byte records. Each file system object requires a header record which stores basic metadata (pathname, owner, permissions, etc.) and zero or more records containing any file data. The end of the archive is indicated by two records consisting entirely of zero bytes.

For compatibility with tape drives that use fixed block sizes, programs that read or write tar files always read or write a fixed number of records with each I/O operation. These "blocks" are always a multiple of the record size. The most common block size—and the maximum supported by historic implementations—is 10240 bytes or 20 records. (Note: the terms "block" and "record" here are not entirely standard; this document follows the convention established by John Gilmore in documenting **pdtar**.)

Old-Style Archive Format

The original tar archive format has been extended many times to include additional information that various implementors found necessary. This section describes the variant implemented by the tar command included in Version 7 AT&T UNIX, which is one of the earliest widely-used versions of the tar program.

The header record for an old-style **tar** archive consists of the following:

```
struct header_old_tar {
        char name[100];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char checksum[8];
        char linkflag[1];
        char linkname[100];
        char pad[255];
};
```

All unused bytes in the header record are filled with nulls.

name Pathname, stored as a null-terminated string. Early tar implementations only stored regular files (including hardlinks to those files). One common early convention used a trailing "/" character to indicate a directory name, allowing directory permissions and owner information to be archived and restored.

mode File mode, stored as an octal number in ASCII.

uid, gid User id and group id of owner, as octal numbers in ASCII.

size Size of file, as octal number in ASCII. For regular files only, this indicates the amount of data that follows the header. In particular, this field was ignored by early tar implementations when extracting hardlinks. Modern writers should always store a zero length for hardlink entries.

mtime Modification time of file, as an octal number in ASCII. This indicates the number of seconds since the start of the epoch, 00:00:00 UTC January 1, 1970. Note that negative values should be avoided here, as they are handled inconsistently.

checksum
: Header checksum, stored as an octal number in ASCII. To compute the checksum, set the checksum field to all spaces, then sum all bytes in the header using unsigned arithmetic. This field should be stored as six octal digits followed by a null and a space character. Note that many early implementations of tar used signed arithmetic for the checksum field, which can cause interoperability problems when transferring archives between systems. Modern robust readers compute the checksum both ways and accept the header if either computation matches.

linkflag, *linkname*
: In order to preserve hardlinks and conserve tape, a file with multiple links is only written to the archive the first time it is encountered. The next time it is encountered, the *linkflag* is set to an ASCII '1' and the *linkname* field holds the first name under which this file appears. (Note that regular files have a null value in the *linkflag* field.)

Early tar implementations varied in how they terminated these fields. The tar command in Version 7 AT&T UNIX used the following conventions (this is also documented in early BSD manpages): the pathname must be null-terminated; the mode, uid, and gid fields must end in a space and a null byte; the size and mtime fields must end in a space; the checksum is terminated by a null and a space. Early implementations filled the numeric fields with leading spaces. This seems to have been common practice until the IEEE Std 1003.1-1988 ("POSIX.1") standard was released. For best portability, modern implementations should fill the numeric fields with leading zeros.

Pre-POSIX Archives

An early draft of IEEE Std 1003.1-1988 ("POSIX.1") served as the basis for John Gilmore's **pdtar** program and many system implementations from the late 1980s and early 1990s. These archives generally follow the POSIX ustar format described below with the following variations:

- The magic value is "ustar " (note the following space). The version field contains a space character followed by a null.
- The numeric fields are generally filled with leading spaces (not leading zeros as recommended in the final standard).
- The prefix field is often not used, limiting pathnames to the 100 characters of old-style archives.

POSIX ustar Archives

IEEE Std 1003.1-1988 ("POSIX.1") defined a standard tar file format to be read and written by compliant implementations of `tar`(1). This format is often called the "ustar" format, after the magic value used in the header. (The name is an acronym for "Unix Standard TAR".) It extends the historic format with new fields:

```
struct header_posix_ustar {
        char name[100];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char checksum[8];
        char typeflag[1];
        char linkname[100];
        char magic[6];
        char version[2];
        char uname[32];
        char gname[32];
        char devmajor[8];
        char devminor[8];
```

```
        char prefix[155];
        char pad[12];
};
```

typeflag Type of entry. POSIX extended the earlier *linkflag* field with several new type values:

- "0" Regular file. NUL should be treated as a synonym, for compatibility purposes.
- "1" Hard link.
- "2" Symbolic link.
- "3" Character device node.
- "4" Block device node.
- "5" Directory.
- "6" FIFO node.
- "7" Reserved.
- Other A POSIX-compliant implementation must treat any unrecognized typeflag value as a regular file. In particular, writers should ensure that all entries have a valid filename so that they can be restored by readers that do not support the corresponding extension. Uppercase letters "A" through "Z" are reserved for custom extensions. Note that sockets and whiteout entries are not archivable.

It is worth noting that the *size* field, in particular, has different meanings depending on the type. For regular files, of course, it indicates the amount of data following the header. For directories, it may be used to indicate the total size of all files in the directory, for use by operating systems that pre-allocate directory space. For all other types, it should be set to zero by writers and ignored by readers.

magic Contains the magic value "ustar" followed by a NUL byte to indicate that this is a POSIX standard archive. Full compliance requires the uname and gname fields be properly set.

version Version. This should be "00" (two copies of the ASCII digit zero) for POSIX standard archives.

uname, *gname*
User and group names, as null-terminated ASCII strings. These should be used in preference to the uid/gid values when they are set and the corresponding names exist on the system.

devmajor, *devminor*
Major and minor numbers for character device or block device entry.

prefix First part of pathname. If the pathname is too long to fit in the 100 bytes provided by the standard format, it can be split at any / character with the first portion going here. If the prefix field is not empty, the reader will prepend the prefix value and a / character to the regular name field to obtain the full pathname.

Note that all unused bytes must be set to NUL.

Field termination is specified slightly differently by POSIX than by previous implementations. The *magic*, *uname*, and *gname* fields must have a trailing NUL. The *pathname*, *linkname*, and *prefix* fields must have a trailing NUL unless they fill the entire field. (In particular, it is possible to store a 256-character pathname if it happens to have a / as the 156th character.) POSIX requires numeric fields to be zero-padded in the front, and allows them to be terminated with either space or NUL characters.

Currently, most tar implementations comply with the ustar format, occasionally extending it by adding new fields to the blank area at the end of the header record.

Pax Interchange Format

There are many attributes that cannot be portably stored in a POSIX ustar archive. IEEE Std 1003.1-2001 ("POSIX.1") defined a "pax interchange format" that uses two new types of entries to hold text-formatted metadata that applies to following entries. Note that a pax interchange format archive is a ustar archive in

every respect. The new data is stored in ustar-compatible archive entries that use the “x” or “g” typeflag. In particular, older implementations that do not fully support these extensions will extract the metadata into regular files, where the metadata can be examined as necessary.

An entry in a pax interchange format archive consists of one or two standard ustar entries, each with its own header and data. The first optional entry stores the extended attributes for the following entry. This optional first entry has an "x" typeflag and a size field that indicates the total size of the extended attributes. The extended attributes themselves are stored as a series of text-format lines encoded in the portable UTF-8 encoding. Each line consists of a decimal number, a space, a key string, an equals sign, a value string, and a new line. The decimal number indicates the length of the entire line, including the initial length field and the trailing newline. An example of such a field is:

```
25 ctime=1084839148.1212\n
```

Keys in all lowercase are standard keys. Vendors can add their own keys by prefixing them with an all uppercase vendor name and a period. Note that, unlike the historic header, numeric values are stored using decimal, not octal. A description of some common keys follows:

atime, **ctime**, **mtime**
: File access, inode change, and modification times. These fields can be negative or include a decimal point and a fractional value.

uname, **uid**, **gname**, **gid**
: User name, group name, and numeric UID and GID values. The user name and group name stored here are encoded in UTF8 and can thus include non-ASCII characters. The UID and GID fields can be of arbitrary length.

linkpath
: The full path of the linked-to file. Note that this is encoded in UTF8 and can thus include non-ASCII characters.

path
: The full pathname of the entry. Note that this is encoded in UTF8 and can thus include non-ASCII characters.

realtime.*, **security.***
: These keys are reserved and may be used for future standardization.

size
: The size of the file. Note that there is no length limit on this field, allowing conforming archives to store files much larger than the historic 8GB limit.

SCHILY.*
: Vendor-specific attributes used by Joerg Schilling’s **star** implementation.

SCHILY.acl.access, **SCHILY.acl.default**
: Stores the access and default ACLs as textual strings in a format that is an extension of the format specified by POSIX.1e draft 17. In particular, each user or group access specification can include a fourth colon-separated field with the numeric UID or GID. This allows ACLs to be restored on systems that may not have complete user or group information available (such as when NIS/YP or LDAP services are temporarily unavailable).

SCHILY.devminor, **SCHILY.devmajor**
: The full minor and major numbers for device nodes.

SCHILY.dev, SCHILY.ino, **SCHILY.nlinks**
: The device number, inode number, and link count for the entry. In particular, note that a pax interchange format archive using Joerg Schilling’s **SCHILY.*** extensions can store all of the data from *struct stat*.

LIBARCHIVE.xattr.*namespace.key*

Libarchive stores POSIX.1e-style extended attributes using keys of this form. The *key* value is URL-encoded: All non-ASCII characters and the two special characters "=" and "%" are encoded as "%" followed by two uppercase hexadecimal digits. The value of this key is the extended attribute value encoded in base 64. XXX Detail the base-64 format here XXX

VENDOR.*

XXX document other vendor-specific extensions XXX

Any values stored in an extended attribute override the corresponding values in the regular tar header. Note that compliant readers should ignore the regular fields when they are overridden. This is important, as existing archivers are known to store non-compliant values in the standard header fields in this situation. There are no limits on length for any of these fields. In particular, numeric fields can be arbitrarily large. All text fields are encoded in UTF8. Compliant writers should store only portable 7-bit ASCII characters in the standard ustar header and use extended attributes whenever a text value contains non-ASCII characters.

In addition to the **x** entry described above, the pax interchange format also supports a **g** entry. The **g** entry is identical in format, but specifies attributes that serve as defaults for all subsequent archive entries. The **g** entry is not widely used.

Besides the new **x** and **g** entries, the pax interchange format has a few other minor variations from the earlier ustar format. The most troubling one is that hardlinks are permitted to have data following them. This allows readers to restore any hardlink to a file without having to rewind the archive to find an earlier entry. However, it creates complications for robust readers, as it is no longer clear whether or not they should ignore the size field for hardlink entries.

GNU Tar Archives

The GNU tar program started with a pre-POSIX format similar to that described earlier and has extended it using several different mechanisms: It added new fields to the empty space in the header (some of which was later used by POSIX for conflicting purposes); it allowed the header to be continued over multiple records; and it defined new entries that modify following entries (similar in principle to the **x** entry described above, but each GNU special entry is single-purpose, unlike the general-purpose **x** entry). As a result, GNU tar archives are not POSIX compatible, although more lenient POSIX-compliant readers can successfully extract most GNU tar archives.

```
struct header_gnu_tar {
        char name[100];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char checksum[8];
        char typeflag[1];
        char linkname[100];
        char magic[6];
        char version[2];
        char uname[32];
        char gname[32];
        char devmajor[8];
        char devminor[8];
        char atime[12];
        char ctime[12];
        char offset[12];
```

```
            char longnames[4];
            char unused[1];
            struct {
                   char offset[12];
                   char numbytes[12];
            } sparse[4];
            char isextended[1];
            char realsize[12];
            char pad[17];
    };
```

typeflag GNU tar uses the following special entry types, in addition to those defined by POSIX:

7 GNU tar treats type "7" records identically to type "0" records, except on one obscure RTOS where they are used to indicate the pre-allocation of a contiguous file on disk.

D This indicates a directory entry. Unlike the POSIX-standard "5" typeflag, the header is followed by data records listing the names of files in this directory. Each name is preceded by an ASCII "Y" if the file is stored in this archive or "N" if the file is not stored in this archive. Each name is terminated with a null, and an extra null marks the end of the name list. The purpose of this entry is to support incremental backups; a program restoring from such an archive may wish to delete files on disk that did not exist in the directory when the archive was made.

Note that the "D" typeflag specifically violates POSIX, which requires that unrecognized typeflags be restored as normal files. In this case, restoring the "D" entry as a file could interfere with subsequent creation of the like-named directory.

K The data for this entry is a long linkname for the following regular entry.

L The data for this entry is a long pathname for the following regular entry.

M This is a continuation of the last file on the previous volume. GNU multi-volume archives guarantee that each volume begins with a valid entry header. To ensure this, a file may be split, with part stored at the end of one volume, and part stored at the beginning of the next volume. The "M" typeflag indicates that this entry continues an existing file. Such entries can only occur as the first or second entry in an archive (the latter only if the first entry is a volume label). The *size* field specifies the size of this entry. The *offset* field at bytes 369-380 specifies the offset where this file fragment begins. The *realsize* field specifies the total size of the file (which must equal *size* plus *offset*). When extracting, GNU tar checks that the header file name is the one it is expecting, that the header offset is in the correct sequence, and that the sum of offset and size is equal to realsize. FreeBSD’s version of GNU tar does not handle the corner case of an archive’s being continued in the middle of a long name or other extension header.

N Type "N" records are no longer generated by GNU tar. They contained a list of files to be renamed or symlinked after extraction; this was originally used to support long names. The contents of this record are a text description of the operations to be done, in the form “Rename %s to %s\n” or “Symlink %s to %s\n”; in either case, both filenames are escaped using K&R C syntax.

S This is a “sparse” regular file. Sparse files are stored as a series of fragments. The header contains a list of fragment offset/length pairs. If more than four such entries are required, the header is extended as necessary with “extra” header extensions (an older format that is no longer used), or “sparse” extensions.

V The *name* field should be interpreted as a tape/volume header name. This entry should generally be ignored on extraction.

magic The magic field holds the five characters "ustar" followed by a space. Note that POSIX ustar archives have a trailing null.

version The version field holds a space character followed by a null. Note that POSIX ustar archives use two copies of the ASCII digit "0".

atime, *ctime*
The time the file was last accessed and the time of last change of file information, stored in octal as with *mtime*.

longnames
This field is apparently no longer used.

Sparse *offset / numbytes*
Each such structure specifies a single fragment of a sparse file. The two fields store values as octal numbers. The fragments are each padded to a multiple of 512 bytes in the archive. On extraction, the list of fragments is collected from the header (including any extension headers), and the data is then read and written to the file at appropriate offsets.

isextended
If this is set to non-zero, the header will be followed by additional "sparse header" records. Each such record contains information about as many as 21 additional sparse blocks as shown here:

```
struct gnu_sparse_header {
        struct {
                char offset[12];
                char numbytes[12];
        } sparse[21];
        char    isextended[1];
        char    padding[7];
};
```

realsize A binary representation of the file's complete size, with a much larger range than the POSIX file size. In particular, with **M** type files, the current entry is only a portion of the file. In that case, the POSIX size field will indicate the size of this entry; the *realsize* field will indicate the total size of the file.

Solaris Tar

XXX More Details Needed XXX

Solaris tar (beginning with SunOS XXX 5.7 ?? XXX) supports an "extended" format that is fundamentally similar to pax interchange format, with the following differences:

- Extended attributes are stored in an entry whose type is **X**, not **x**, as used by pax interchange format. The detailed format of this entry appears to be the same as detailed above for the **x** entry.
- An additional **A** entry is used to store an ACL for the following regular entry. The body of this entry contains a seven-digit octal number (whose value is 01000000 plus the number of ACL entries) followed by a zero byte, followed by the textual ACL description.

Other Extensions

One common extension, utilized by GNU tar, star, and other newer **tar** implementations, permits binary numbers in the standard numeric fields. This is flagged by setting the high bit of the first character. This permits 95-bit values for the length and time fields and 63-bit values for the uid, gid, and device numbers. GNU tar supports this extension for the length, mtime, ctime, and atime fields. Joerg Schilling's star program sup-

ports this extension for all numeric fields. Note that this extension is largely obsoleted by the extended attribute record provided by the pax interchange format.

Another early GNU extension allowed base-64 values rather than octal. This extension was short-lived and such archives are almost never seen. However, there is still code in GNU tar to support them; this code is responsible for a very cryptic warning message that is sometimes seen when GNU tar encounters a damaged archive.

SEE ALSO

ar(1), pax(1), tar(1)

STANDARDS

The **tar** utility is no longer a part of POSIX or the Single Unix Standard. It last appeared in Version 2 of the Single UNIX Specification ("SUSv2"). It has been supplanted in subsequent standards by pax(1). The ustar format is currently part of the specification for the pax(1) utility. The pax interchange file format is new with IEEE Std 1003.1-2001 ("POSIX.1").

HISTORY

A **tar** command appeared in Seventh Edition Unix, which was released in January, 1979. It replaced the **tp** program from Fourth Edition Unix which in turn replaced the **tap** program from First Edition Unix. John Gilmore's **pdtar** public-domain implementation (circa 1987) was highly influential and formed the basis of **GNU tar**. Joerg Shilling's **star** archiver is another open-source (GPL) archiver (originally developed circa 1985) which features complete support for pax interchange format.

NAME

targets — configuration file for iSCSI targets

SYNOPSIS

targets

DESCRIPTION

The **targets** file describes the iSCSI storage which is presented to iSCSI initiators by the iscsi-target(8) service. A description of the iSCSI protocol can be found in *Internet Small Computer Systems Interface RFC 3720*.

Each line in the file (other than comment lines that begin with a '#') specifies an extent, a device (made up of extents or other devices), or a target to present to the initiator.

Each definition, an extent, a device, and a target, is specified on a single whitespace delimited line.

The *extent* definition specifies a piece of storage that will be used as storage, and presented to initiators. It is the basic definition for an iSCSI target. Each target must contain at least one extent definition. The first field in the definition is the extent name, which must begin with the word "extent" and be followed by a number. The next field is the file or NetBSD device which will be used as persistent storage. The next field is the offset (in bytes) of the start of the extent. This field is usually 0. The fourth field in the definition is the size of the extent. The basic unit is bytes, and the shorthand *KB*, *MB*, *GB*, and *TB* can be used for kilobytes (1024 bytes), megabytes (1024 kilobytes), gigabytes (1024 megabytes), and terabytes (1024 gigabytes) respectively. It is possible to use the word "size" to use the full size of the pre-existing regular file given in the extent name.

The *device* definition specifies a LUN or device, and is made up of extents and other devices. It is possible to create hierarchies of devices using the device definition. The first field in the definition is the device name, which must begin with the word "device" and be followed by a number. The next field is the type of resilience that is to be provided by the device. For simple devices, *RAID0* suffices. Greater resilience can be gained by using the *RAID1* resilience field. Following the resilience field is a list of extents or other devices. Large devices can be created by using multiple RAID0 extents, in which case each extent will be concatenated. Resilient devices can be created by using multiple RAID1 devices or extents, in which case data will be written to each of the devices or extents in turn. If RAID1 resilience is used, all the extents or sub-devices must be the same size. Please note that RAID1 recovery is not yet supported by the iscsi-target(8) utility. An extent or sub-device may only be used once.

The *target* definition specifies an iSCSI target, which is presented to the iSCSI initiator. Multiple targets can be specified. The first field in the definition is the target name, which must begin with either of the words "target" or "lun" and be followed by a number. Optionally, if a target is followed by an "=" sign and some text, the text is taken to be that of the iSCSI Qualified Name of the target. This IQN is used by the initiator to connect to the appropriate target. The next field is a selector for whether the storage should be presented as writable, or merely as read-only storage. The field of "rw" denotes read-write storage, whilst "ro" denotes read-only storage. The next field is the device or extent name that will be used as persistent storage for this target. The fourth field is a slash-notation netmask which will be used, during the discovery phase, to control the network addresses to which targets will be presented. The magic values "any" and "all" will expand to be the same as "0/0". If an attempt is made to discover a target which is not allowed by the netmask, a warning will be issued using syslog(3) to make administrators aware of this attempt. The administrator can still use tcp wrapper functionality, as found in hosts_access(5) and hosts.deny(5) to allow or deny discovery attempts from initiators as well as using the inbuilt netmask functionality.

FILES

`/etc/iscsi/targets` the list of exported storage **`targets`**

SEE ALSO

`syslog`(3), `hosts_access`(5), `hosts.deny`(5), `iscsi-target`(8)

HISTORY

The **`targets`** file first appeared in NetBSD 4.0.

NAME

tcp_table − Postfix client/server table lookup protocol

SYNOPSIS

postmap -q "*string***" tcp:***host:port*

postmap -q - tcp:*host:port* <*inputfile*

DESCRIPTION

The Postfix mail system uses optional tables for address rewriting or mail routing. These tables are usually in **dbm** or **db** format. Alternatively, table lookups can be directed to a TCP server.

To find out what types of lookup tables your Postfix system supports use the "**postconf -m**" command.

To test lookup tables, use the "**postmap -q**" command as described in the SYNOPSIS above.

PROTOCOL DESCRIPTION

The TCP map class implements a very simple protocol: the client sends a request, and the server sends one reply. Requests and replies are sent as one line of ASCII text, terminated by the ASCII newline character. Request and reply parameters (see below) are separated by whitespace.

Send and receive operations must complete in 100 seconds.

REQUEST FORMAT

Each request specifies a command, a lookup key, and possibly a lookup result.

get SPACE *key* NEWLINE
: Look up data under the specified key.

put SPACE *key* SPACE *value* NEWLINE
: This request is currently not implemented.

REPLY FORMAT

Each reply specifies a status code and text. Replies must be no longer than 4096 characters including the newline terminator.

500 SPACE *text* NEWLINE
: In case of a lookup request, the requested data does not exist. In case of an update request, the request was rejected. The text describes the nature of the problem.

400 SPACE *text* NEWLINE
: This indicates an error condition. The text describes the nature of the problem. The client should retry the request later.

200 SPACE *text* NEWLINE
: The request was successful. In the case of a lookup request, the text contains an encoded version of the requested data.

ENCODING

In request and reply parameters, the character %, each non-printing character, and each whitespace character must be replaced by %XX, where XX is the corresponding ASCII hexadecimal character value. The hexadecimal codes can be specified in any case (upper, lower, mixed).

The Postfix client always encodes a request. The server may omit the encoding as long as the reply is guaranteed to not contain the % or NEWLINE character.

SECURITY

Do not use TCP lookup tables for security critical purposes. The client-server connection is not protected and the server is not authenticated.

BUGS

Only the lookup method is currently implemented.

The client does not hang up when the connection is idle for a long time.

SEE ALSO

postmap(1), Postfix lookup table manager
regexp_table(5), format of regular expression tables
pcre_table(5), format of PCRE tables
cidr_table(5), format of CIDR tables

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
DATABASE_README, Postfix lookup table overview

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

termcap — terminal capability data base

SYNOPSIS

termcap

DESCRIPTION

The **termcap** file is a database describing terminals, used, for example, by vi(1) and curses(3). Terminals are described in **termcap** by giving a set of capabilities that they have and by describing how operations are performed. Padding requirements and initialization sequences are included in **termcap**.

Entries in **termcap** consist of a number of ':'-separated fields. The first entry for each terminal gives the names that are known for the terminal, separated by '|' characters. The first name given is the most common abbreviation for the terminal. The last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lower case and contain no blanks; the last name may well contain upper case characters and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, thus "hp2621" This name should not contain hyphens. Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Therefore, a "vt100" in 132-column mode would be "vt100-w". The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With automatic margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
−*n*	Number of lines on screen	aaa-60
-na	No arrow keys (leave them in local)	concept100-na
−*np*	Number of pages of memory	concept100-4p
-rv	Reverse video	concept100-rv

CAPABILITIES

The characters in the *Notes* function field in the table have the following meanings (more than one may apply to a capability):

N	indicates numeric parameter(s)
P	indicates that padding may be specified
*	indicates that padding may be based on the number of lines affected
o	indicates capability is obsolete

"Obsolete" capabilities have no *terminfo* equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them at all.

Name	Type	Notes	Description
functions			
!1	str		Sent by shifted save key.
!2	str		Sent by shifted suspend key.
!3	str		Sent by shifted undo key.
#1	str		Sent by shifted help key.
#2	str		Sent by shifted home key.
#3	str		Sent by shifted input key.

#4	str	Sent by shifted left-arrow key.
%0	str	Sent by redo key.
%1	str	Sent by help key.
%2	str	Sent by mark key.
%3	str	Sent by message key.
%4	str	Sent by move key.
%5	str	Sent by next-object key.
%6	str	Sent by open key.
%7	str	Sent by options key.
%8	str	Sent by previous-object key.
%9	str	Sent by print or copy key.
%a	str	Sent by shifted message key.
%b	str	Sent by shifted move key.
%c	str	Sent by shifted next key.
%d	str	Sent by shifted options key.
%e	str	Sent by shifted prev key.
%f	str	Sent by shifted print key.
%g	str	Sent by shifted redo key.
%h	str	Sent by shifted replace key.
%i	str	Sent by shifted right-arrow key.
%j	str	Sent by shifted resume key.
&0	str	Sent by shifted cancel key.
&1	str	Sent by ref(erence) key.
&2	str	Sent by refresh key.
&3	str	Sent by replace key.
&4	str	Sent by restart key.
&5	str	Sent by resume key.
&6	str	Sent by save key.
&7	str	Sent by suspend key.
&8	str	Sent by undo key.
&9	str	Sent by shifted begin key.
∗0	str	Sent by shifted find key.
∗1	str	Sent by shifted command key.
∗2	str	Sent by shifted copy key.
∗3	str	Sent by shifted create key.
∗4	str	Sent by shifted delete-character key.
∗5	str	Sent by shifted delete-line key.
∗6	str	Sent by shifted select key.
∗7	str	Sent by shifted end key.
∗8	str	Sent by shifted clear-to-end-of-line key.
∗9	str	Sent by shifted exit key.
@0	str	Sent by find key.
@1	str	Sent by begin key.
@2	str	Sent by cancel key.
@3	str	Sent by close key.
@4	str	Sent by command key.
@5	str	Sent by copy key.
@6	str	Sent by create key.
@7	str	Sent by end key.

@8	str		Sent by enter key.
@9	str		Sent by exit key.
AB	str		Set background color to #1 using ANSI escape.
ac	str		Alternative (graphic) character set pairs.
ae	str	(P)	End alternative character set.
AL	str	(NP∗)	Add" *n* new blank lines
AF	str		Set foreground color to #1 using ANSI escape.
al	str	(P∗)	Add new blank line.
am	bool		Terminal has automatic margins.
as	str	(P)	Start alternative character set.
bc	str	(o)	Backspace if not **^H**.
bl	str	(P)	Audible signal (bell).
bs	bool	(o)	Terminal can backspace with **^H**.
bt	str	(P)	Back tab.
bw	bool		**le** (backspace) wraps from column 0 to last column.
CC	str		Terminal settable command character in prototype.
cc	bool		Terminal can re-define existing color.
cd	str	(P∗)	Clear to end of display.
ce	str	(P)	Clear to end of line.
ch	str	(NP)	Set cursor column (horizontal position).
cl	str	(P∗)	Clear screen and home cursor.
CM	str	(NP)	Memory-relative cursor addressing.
cm	str	(NP)	Screen-relative cursor motion.
Co	num		Maximum number of colors on the screen.
co	num		Number of columns in a line (See **BUGS** section below).
cr	str	(P)	Carriage return.
cs	str	(NP)	Change scrolling region (VT100).
ct	str	(P)	Clear all tab stops.
cv	str	(NP)	Set cursor row (vertical position).
da	bool		Display may be retained above the screen.
dB	num	(o)	Milliseconds of **bs** delay needed (default 0).
db	bool		Display may be retained below the screen.
DC	str	(NP∗)	Delete *n* characters.
dC	num	(o)	Milliseconds of **cr** delay needed (default 0).
dc	str	(P∗)	Delete character.
dF	num	(o)	Milliseconds of **ff** delay needed (default 0).
DL	str	(NP∗)	Delete *n* lines.
dl	str	(P∗)	Delete line.
dm	str		Enter delete mode.
dN	num	(o)	Milliseconds of **nl** delay needed (default 0).
DO	str	(NP∗)	Move cursor down: *n* lines.
do	str		Down one line.
ds	str		Disable status line.
dT	num	(o)	Milliseconds of horizontal tab delay needed (default 0).
dV	num	(o)	Milliseconds of vertical tab delay needed (default 0).
eA	str		Enable alternate character set.
ec	str	(NP)	Erase *n* characters.
ed	str		End delete mode.
ei	str		End insert mode.

eo	bool		Can erase overstrikes with a blank.
EP	bool	(o)	Even parity.
es	bool		Escape can be used on the status line.
F1-F9	str		Sent by function keys 11-19.
FA-FZ	str		Sent by function keys 20-45.
Fa-Fr	str		Sent by function keys 46-63.
ff	str	(P∗)	Hardcopy terminal page eject.
fs	str		Return from status line.
gn	bool		Generic line type, for example dialup, switch).
hc	bool		Hardcopy terminal.
HD	bool	(o)	Half-duplex.
hd	str		Half-line down (forward 1/2 linefeed).
hl	bool		Terminal uses only HLS color notation (Tektronix).
ho	str	(P)	Home cursor.
hs	bool		Has extra “status line”.
hu	str		Half-line up (reverse 1/2 linefeed).
hz	bool		Cannot print “~” (Hazeltine).
i1-i3	str		Terminal initialization strings (`terminfo`(5) only)
IC	str	(NP∗)	Insert *n* blank characters.
Ic	str		Set color #1 to RGB #2, #3, #4.
ic	str	(P∗)	Insert character.
if	str		Name of file containing initialization string.
im	str		Enter insert mode.
in	bool		Insert mode distinguishes nulls.
Ip	str		Set color-pair #1 to fg #2, bg #3.
iP	str		Pathname of program for initialization (`terminfo`(5) only).
ip	str	(P∗)	Insert pad after character inserted.
is	str		Terminal initialization string (**`termcap`** only).
it	num		Tabs initially every *n* positions.
K1	str		Sent by keypad upper left.
K2	str		Sent by keypad center.
K3	str		Sent by keypad upper right.
K4	str		Sent by keypad lower left.
K5	str		Sent by keypad lower right.
k0-k9	str		Sent by function keys 0-9.
k;	str		Sent by function key 10.
kA	str		Sent by insert-line key.
ka	str		Sent by clear-all-tabs key.
kB	str		Sent by back-tab key.
kb	str		Sent by backspace key.
kC	str		Sent by clear-screen or erase key.
kD	str		Sent by delete-character key.
kd	str		Sent by down-arrow key.
kE	str		Sent by clear-to-end-of-line key.
ke	str		Out of “keypad transmit” mode.
kF	str		Sent by scroll-forward/down key.
kH	str		Sent by home-down key.
kh	str		Sent by home key.
kI	str		Sent by insert-character or enter-insert-mode key.

kL	str		Sent by delete-line key.
kl	str		Sent by left-arrow key.
kM	str		Sent by insert key while in insert mode.
Km	str		Mouse event has occurred.
km	bool		Has a “meta” key (shift, sets parity bit).
kN	str		Sent by next-page key.
kn	num	(o)	Number of function (**k0**− **k9**) keys (default 0).
ko	str	(o)	Termcap entries for other non-function keys.
kP	str		Sent by previous-page key.
kR	str		Sent by scroll-backward/up key.
kr	str		Sent by right-arrow key.
kS	str		Sent by clear-to-end-of-screen key.
ks	str		Put terminal in “keypad transmit” mode.
kT	str		Sent by set-tab key.
kt	str		Sent by clear-tab key.
ku	str		Sent by up-arrow key.
l0-l9	str		Labels on function keys if not “f*n*”.
LC	bool	(o)	Lower-case only.
LE	str	(NP)	Move cursor left *n* positions.
le	str	(P)	Move cursor left one position.
li	num		Number of lines on screen or page (See **BUGS** section below)
ll	str		Last line, first column
lm	num		Lines of memory if > **li** (0 means varies).
ma	str	(o)	Arrow key map (used by `vi`(1) version 2 only).
mb	str		Turn on blinking attribute.
md	str		Turn on bold (extra bright) attribute.
me	str		Turn off all attributes.
mh	str		Turn on half-bright attribute.
mi	bool		Safe to move while in insert mode.
mk	str		Turn on blank attribute (characters invisible).
ml	str	(o)	Memory lock on above cursor.
mm	str		Turn on “meta mode” (8th bit).
mo	str		Turn off “meta mode”.
mp	str		Turn on protected attribute.
mr	str		Turn on reverse-video attribute.
ms	bool		Safe to move in standout modes.
mu	str	(o)	Memory unlock (turn off memory lock).
NC	num		Video attributes that can’t be used with colors
nc	bool	(o)	No correctly-working **cr** (Datamedia 2500, Hazeltine 2000).
nd	str		Non-destructive space (cursor right).
NL	bool	(o)	**\n** is newline, not line feed.
nl	str	(o)	Newline character if not **\n**.
ns	bool	(o)	Terminal is a CRT but doesn’t scroll.
nw	str	(P)	Newline (behaves like **cr** followed by **do**).
OP	bool	(o)	Odd parity.
oc	str		Set all colors to the original ones.
op	str		Set default color-pair to the original one.
os	bool		Terminal overstrikes.
pa	num		Maximum number of color-pairs on the screen.

pb	num		Lowest baud where delays are required.
pc	str		Pad character (default NUL).
pf	str		Turn off the printer.
pk	str		Program function key *n* to type string *s* (`terminfo`(5) only).
pl	str		Program function key *n* to execute string *s* (`terminfo`(5) only).
pO	str	(N)	Turn on the printer for *n* bytes.
po	str		Turn on the printer.
ps	str		Print contents of the screen.
pt	bool	(o)	Has hardware tabs (may need to be set with **is**).
px	str		Program function key *n* to transmit string *s* (`terminfo`(5) only).
r1-r3	str		Reset terminal completely to sane modes (`terminfo`(5) only).
rc	str	(P)	Restore cursor to position of last **sc**.
rf	str		Name of file containing reset codes.
RI	str	(NP)	Move cursor right *n* positions.
rp	str	(NP∗)	Repeat character *c* *n* times.
rs	str		Reset terminal completely to sane modes (**`termcap`** only).
sa	str	(NP)	Define the video attributes.
Sb	str		Set background color to #1.
sc	str	(P)	Save cursor position.
se	str		End standout mode.
Sf	str		Set foreground color to #1.
SF	str	(NP∗)	Scroll forward *n* lines.
sf	str	(P)	Scroll text up.
sg	num		Number of garbage chars left by **so** or **se** (default 0).
so	str		Begin standout mode.
SR	str	(NP∗)	Scroll backward *n* lines.
sr	str	(P)	Scroll text down.
st	str		Set a tab in all rows, current column.
ta	str	(P)	Tab to next 8-position hardware tab stop.
tc	str		Entry of similar terminal − must be last.
te	str		String to end programs that use **`termcap`**.
ti	str		String to begin programs that use **`termcap`**.
ts	str	(N)	Go to status line, column *n*.
UC	bool	(o)	Upper-case only.
uc	str		Underscore one character and move past it.
ue	str		End underscore mode.
ug	num		Number of garbage chars left by **us** or **ue** (default 0).
ul	bool		Underline character overstrikes.
UP	str	(NP∗)	Move cursor up *n* lines.
up	str		Upline (cursor up).
us	str		Start underscore mode.
ut	bool		Screen erased with background color.
vb	str		Visible bell (must not move cursor).
ve	str		Make cursor appear normal (undo **vs**/ **vi**).
vi	str		Make cursor invisible.
vs	str		Make cursor very visible.
vt	num		Virtual terminal number (not supported on all systems).
wi	str	(N)	Set current window.
ws	num		Number of columns in status line.

xb	bool		Beehive (f1= ESC, f2=**^C**).
xn	bool		Newline ignored after 80 cols (Concept).
xo	bool		Terminal uses xoff/xon (DC3/DC1) handshaking.
xr	bool	(o)	Return acts like **ce cr nl** (Delta Data).
xs	bool		Standout not erased by overwriting (Hewlett-Packard).
xt	bool		Tabs ruin, magic **so** char (Teleray 1061).
xx	bool	(o)	Tektronix 4025 insert-line.
ZZ	str		Pointer to the extended termcap entry. See tgetent(3) for details.

A Sample Entry

The following entry, which describes the Concept−100, is among the more complex entries in the **termcap** file as of this writing.

```
ca|concept100|c100|concept|c104|concept100-4p|HDS Concept−100:\
        :al=3*\E^R:am:bl=^G:cd=16*\E^C:ce=16\E^U:cl=2*^L:cm=\Ea%+ %+ :\
        :co#80:.cr=9^M:db:dc=16\E^A:dl=3*\E^B:do=^J:ei=\E\200:eo:im=\E^P:in:\
        :ip=16*:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E:k1=\E5:\
        :k2=\E6:k3=\E7:kb=^h:kd=\E<:ke=\Ex:kh=\E?:kl=\E>:kr=\E=:ks=\EX:\
        :ku=\E;:le=^H:li#24:mb=\EC:me=\EN\200:mh=\EE:mi:mk=\EH:mp=\EI:\
        :mr=\ED:nd=\E=:pb#9600:rp=0.2*\Er%.%+ :se=\Ed\Ee:sf=^J:so=\EE\ED:\
        :.ta=8\t:te=\Ev    \200\200\200\200\200\200\Ep\r\n:\
        :ti=\EU\Ev  8p\Ep\r:ue=\Eg:ul:up=\E;:us=\EG:\
        :vb=\Ek\200\200\200\200\200\200\200\200\200\200\200\200\200\EK:\
        :ve=\Ew:vs=\EW:vt#8:xn:\
        :bs:cr=^M:dC#9:dT#8:nl=^J:ta=^I:pt:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with "#".

Types of Capabilities

Capabilities in **termcap** are of three types: Boolean capabilities, which indicate particular features that the terminal has; numeric capabilities, giving the size of the display or the size of other attributes; and string capabilities, which give character sequences that can be used to perform particular terminal operations. All capabilities have two-letter codes. For instance, the fact that the Concept has *automatic margins* (an automatic return and linefeed when the end of a line is reached) is indicated by the Boolean capability **am**. Hence the description of the Concept includes **am**.

Numeric capabilities are followed by the character '#' then the value. In the example above **co**, which indicates the number of columns the display has, gives the value '80' for the Concept.

Finally, string-valued capabilities, such as **ce** (clear-to-end-of-line sequence) are given by the two-letter code, an '=', then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, which causes padding characters to be supplied by tputs(3) after the remainder of the string is sent to provide this delay. The delay can be either a number, such as '20', or a number followed by an '*', such as '3*'. An '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-line padding required. (In the case of insert-character, the factor is still the number of *lines* affected; this is always 1 unless the terminal has **in** and the software uses it.) When an '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of control characters there. **\E** maps to an ESC character, **^X** maps to a control-X for any appropriate X, and the sequences **\n \r \t \b \f** map to linefeed, return, tab, backspace, and formfeed, respectively. Finally, characters

may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a **:** in a capability it must be escaped in octal as **\072**. If it is necessary to place a NUL character in a string capability it must be encoded as **\200**. (The routines that deal with **termcap** use C strings and strip the high bits of the output very late, so that a **\200** comes out as a **\000** would.)

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the first **cr** and **ta** in the example above.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **termcap** and to build up a description gradually, using partial descriptions with vi(1) to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **termcap** file to describe it or bugs in vi(1). To easily test a new terminal description you are working on you can put it in your home directory in a file called .termcap and programs will look there before looking in /usr/share/misc/termcap. You can also set the environment variable TERMPATH to a list of absolute file pathnames (separated by spaces or colons), one of which contains the description you are working on, and programs will search them in the order listed, and nowhere else. See termcap(3). The TERMCAP environment variable is usually set to the **termcap** entry itself to avoid reading files when starting up a program.

To get the padding for insert-line right (if the terminal manufacturer did not document it), a severe test is to use vi(1) to edit /etc/passwd at 9600 baud, delete roughly 16 lines from the middle of the screen, then hit the 'u' key several times quickly. If the display messes up, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line of the display is given by the **co** numeric capability. If the display is a CRT, then the number of lines on the screen is given by the **li** capability. If the display wraps around to the beginning of the next line when the cursor reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, the code to do this is given by the **cl** string capability. If the terminal overstrikes (rather than clearing the position when a character is overwritten), it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as the Tektronix 4010 series, as well as to hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage-return, **^M**.) If there is a code to produce an audible signal (bell, beep, etc.), give this as **bl**.

If there is a code (such as backspace) to move the cursor one position to the left, that capability should be given as **le**. Similarly, codes to move to the right, up, and down should be given as **nd**, **up**, and **do**, respectively. These *local cursor motions* should not alter the text they pass over; for example, you would not normally use "nd= " unless the terminal has the **os** capability, because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **termcap** have undefined behavior at the left and top edges of a CRT display. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up off the top using local cursor motions.

In order to scroll text up, a program goes to the bottom left corner of the screen and sends the **sf** (index) string. To scroll text down, a program goes to the top left corner of the screen and sends the **sr** (reverse index) string. The strings **sf** and **sr** have undefined behavior when not on their respective corners of the screen. Parameterized versions of the scrolling sequences are **SF** and **SR**, which have the same semantics as **sf** and **sr** except that they take one parameter and scroll that many lines. They also have undefined behavior except at the appropriate corner of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output there, but this does not necessarily apply to **nd** from the last column. Leftward local motion is defined from the left edge only when **bw** is given; then an **le** from the left edge will move to the right edge of the previous row. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the **termcap** description usually assumes that this feature is on, i.e., **am**. If the terminal has a command that moves to the first column of the next line, that command can be given as **nw** (newline). It is permissible for this to clear the remainder of the current line, so if the terminal has no correctly-working CR and LF it may still be possible to craft a working **nw** out of one or both of them.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the Teletype model 33 is described as

```
T3|tty33|33|tty|Teletype model 33:\
        :bl=^G:co#72:cr=^M:do=^J:hc:os:
```

and the Lear Siegler ADM−3 is described as

```
l3|adm3|3|LSI ADM-3:\
:am:bl=^G:cl=^Z:co#80:cr=^M:do=^J:le=^H:li#24:sf=^J:
```

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with printf(3)−like escapes **%x** in it, while other characters are passed through unchanged. For example, to address the cursor the **cm** capability is given, using two parameters: the row and column to move to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory. If the terminal has memory-relative cursor addressing, that can be indicated by an analogous **CM** capability.)

The **%** encodings have the following meanings:

%%	output '%'
%d	output value as in printf(3) %d
%2	output value as in printf(3) %2d
%3	output value as in printf(3) %3d
%.	output value as in printf(3) %c
%+*x*	add *x* to value, then do %.
%>*xy*	if value > *x* then add *y*, no output
%r	reverse order of two parameters, no output
%i	increment by one, no output
%n	exclusive-or all parameters with 0140 (Datamedia 2500)
%B	BCD (16∗(value/10)) + (value%10), no output
%D	Reverse coding (value − 2∗(value%16)), no output (Delta Data).

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent "\E&a12c03Y" padded for 6 milliseconds. Note that the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its **cm** capability is "cm=6\E&%r%2c%2Y".

The Datamedia 2500 needs the current row and column sent encoded in binary using "%.". Terminals that use "%." need to be able to backspace the cursor (**le**) and to move the cursor up one line on the screen (**up**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (Programs using **termcap** must set terminal modes so that tabs are not expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM−3a, which offsets row and column by a blank character, thus "cm=\E=%+ %+ ".

Row or column absolute cursor addressing can be given as single parameter capabilities **ch** (horizontal position absolute) and **cv** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cm**. If there are parameterized local motions (e.g., move *n* positions to the right) these can be given as **DO**, **LE**, **RI**, and **UP** with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have **cm**, such as the Tektronix 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to the very upper left corner of the screen), this can be given as **ho**. Similarly, a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **up** from the home position, but a program should never do this itself (unless **ll** does), because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as cursor address (0,0): to the top left corner of the screen, not of memory. (Therefore, the "\EH" sequence on Hewlett-Packard terminals cannot be used for **ho**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, this should be given as **cd**. **cd** must only be invoked from the first column of a line. (Therefore, it can be simulated by a request to delete a large number of lines, if a true **cd** is not available.)

Insert/Delete Line

If the terminal can open a new blank line before the line containing the cursor, this should be given as **al**; this must be invoked only from the first position of a line. The cursor must then appear at the left of the newly blank line. If the terminal can delete the line that the cursor is on, this should be given as **dl**; this must only be used from the first position on the line to be deleted. Versions of **al** and **dl** which take a single parameter and insert or delete that many lines can be given as **AL** and **DL**. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the **cs** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **sr** or **sf** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory which all commands affect, it should be given as the parameterized string **wi**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order. (This `terminfo`(5) capability is described for completeness. It is unlikely that any **`termcap`**−using program will support it.)

If the terminal can retain display memory above the screen, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **sr** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using **`termcap`**. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept−100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen then typing text separated by cursor motions. Type "`abc  def`" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distin-

guish between blanks and untyped positions. If the “abc” shifts over to the “def” which then move together around the end of the current line and onto the next as you insert, then you have the second type of terminal and should give the capability **in**, which stands for “insert null”. While these are two logically separate attributes (one-line vs. multi-line insert mode, and special treatment of untyped spaces), we have seen no terminals whose insert mode cannot be described with the single attribute.

termcap can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode. Give as **ei** the sequence to leave insert mode. Now give as **ic** any sequence that needs to be sent just before each character to be inserted. Most terminals with a true insert mode will not give **ic**; terminals that use a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ic**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after insertion of a single character can also be given in **ip**. If your terminal needs to be placed into an ‘insert mode’ and needs a special code preceding each inserted character, then both **im**/ **ei** and **ic** can be given, and both will be used. The **IC** capability, with one parameter *n*, will repeat the effects of **ic** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia’s) must not have **mi** because of the way their insert mode works.

Finally, you can specify **dc** to delete a single character, **DC** with one parameter *n* to delete *n* characters, and delete mode by giving **dm** and **ed** to enter and exit delete mode (which is any mode the terminal needs to be placed in for **dc** to work).

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode*, representing a good high-contrast, easy-on-the-eyes format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **so** and **se**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces or garbage characters on the screen, as the TVI 912 and Teleray 1061 do, then **sg** should be given to tell how many characters are left.

Codes to begin underlining and end underlining can be given as **us** and **ue**, respectively. Underline mode change garbage is specified by **ug**, similar to **sg**. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **mb** (blinking), **md** (bold or extra bright), **mh** (dim or half-bright), **mk** (blanking or invisible text), **mp** (protected), **mr** (reverse video), **me** (turn off *all* attribute modes), **as** (enter alternative character set mode), and **ae** (exit alternative character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of mode, this should be given as **sa** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attributes is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternative character set. Not all modes need be supported by **sa**, only those for which corresponding attribute commands exist. (It is unlikely that a **termcap**−using program will support this capability, which is defined for compatibility with terminfo(5)).

Terminals with the “magic cookie” glitches (**sg** and **ug**), rather than maintaining extra attribute bits for each character cell, instead deposit special “cookies”, or “garbage characters”, when they receive mode-setting sequences, which affect the display algorithm.

Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or when the cursor is addressed. Programs using standout mode should exit standout mode on such terminals before moving the cursor or sending a newline. On terminals where this is not a problem, the **ms** capability should be present to say that this overhead is unnecessary.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as **vb**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to change, for example, a non-blinking underline into an easier-to-find block or blinking underline), give this sequence as **vs**. If there is a way to make the cursor completely invisible, give that as **vi**. The capability **ve**, which undoes the effects of both of these modes, should also be given.

If your terminal correctly displays underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local mode (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left-arrow, right-arrow, up-arrow, down-arrow, and home keys can be given as **kl**, **kr**, **ku**, **kd**, and **kh**, respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0**, **k1**, **k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0**, **l1**, **l9**. The codes transmitted by certain other special keys can be given: **kH** (home down), **kb** (backspace), **ka** (clear all tabs), **kt** (clear the tab stop in this column), **kC** (clear screen or erase), **kD** (delete character), **kL** (delete line), **kM** (exit insert mode), **kE** (clear to end of line), **kS** (clear to end of screen), **kI** (insert character or enter insert mode), **kA** (insert line), **kN** (next page), **kP** (previous page), **kF** (scroll forward/down), **kR** (scroll backward/up), and **kT** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, then the other five keys can be given as **K1**, **K2**, **K3**, **K4**, and **K5**. These keys are useful when the effects of a 3 by 3 directional pad are needed. The obsolete **ko** capability formerly used to describe "other" function keys has been completely supplanted by the above capabilities.

The **ma** entry is also used to indicate arrow keys on terminals that have single-character arrow keys. It is obsolete but still in use in version 2 of **vi** which must be run on some minicomputers due to memory limitations. This field is redundant with **kl**, **kr**, **ku**, **kd**, and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, and the second character is the corresponding **vi** command. These commands are *h* for **kl**, *j* for **kd**, *k* for **ku**, *l* for **kr**, and *H* for **kh**. For example, the Mime would have "`ma=^Hh^Kj^Zk^Xl`" indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Tabs and Initialization

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory-relative cursor addressing and not screen-relative cursor addressing, a screen-sized window must be fixed into the display for cursor addressing to work properly. This is also used for the Tektronix 4025, where **ti** sets the command character to be the one used by **`termcap`**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **`termcap`** description. They are normally sent to the terminal by the `tset`(1) program each time the user logs in. They will be printed in the following order: **is**; setting tabs using **ct** and **st**; and finally **if**.

(`terminfo`(5) uses **i1-i2** instead of **is** and runs the program **iP** and prints **i3** after the other initializations.) A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs** and **if**. These strings are output by the `reset`(1) program, which is used when the terminal gets into a wedged state. (`terminfo`(5) uses **r1-r3** instead of **rs**.) Commands are normally placed in **rs** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the VT100 into 80-column mode would normally be part of **is**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80-column mode.

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ta** (usually **^I**). A "backtab" command which moves leftward to the previous tab stop can be given as **bt**. By convention, if the terminal driver modes indicate that tab stops are being expanded by the computer rather than being sent to the terminal, programs should not use **ta** or **bt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every *n* positions when the terminal is powered up, then the numeric parameter **it** is given, showing the number of positions between tab stops. This is normally used by the `tset`(1) command to determine whether to set the driver mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **`termcap`** description can assume that they are properly set.

If there are commands to set and clear tab stops, they can be given as **ct** (clear all tab stops) and **st** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is** or **if**.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the `tset`(1) program to set terminal driver modes appropriately. Delays embedded in the capabilities **cr**, **sf**, **le**, **ff**, and **ta** will cause the appropriate delay bits to be set in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**. For 4.2BSD `tset`(1), the delays are given as numeric capabilities **dC**, **dN**, **dB**, **dF**, and **dT** instead.

Miscellaneous

If the terminal requires other than a `NUL` (zero) character as a pad, this can be given as **pc**. Only the first character of the **pc** string is used.

If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**.

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, then the capability **hs** should be given. Special strings to go to a position in the status line and to return from the status line can be given as **ts** and **fs**. (**fs** must leave the cursor position in the same place that it was before **ts**. If necessary, the **sc** and **rc** strings can be included in **ts** and **fs** to get this effect.) The capability **ts** takes one parameter, which is the column number of the status line to which the cursor is to be moved. If escape sequences and other special commands such as tab work while in the status line, the flag **es** can be given. A string that turns off the status line (or otherwise erases its contents) should be given as **ds**. The status line is normally assumed to be the same width as the rest of the screen, i.e., **co**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), then its width in columns can be indicated with the numeric parameter **ws**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually **^L**).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters), this can be indicated with the parameterized string **rp**. The first parameter is the character to be repeated and the second is the number of times to repeat it. (This is a `terminfo`(5) feature that is unlikely to be supported by a program that uses **`termcap`**.)

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **CC**. A prototype command character is chosen which is used in all capabilities. This character is given in the **CC** capability to identify it. The following convention is supported on some UNIX systems: The environment is to be searched for a `CC` variable, and if found, all occurrences of the prototype character are replaced by the character in the environment variable. This use of the `CC` environment variable is a very bad idea, as it conflicts with `make`(1).

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.)

If the terminal uses xoff/xon (DC3/DC1) handshaking for flow control, give **xo**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, then this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **mm** and **mo**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. An explicit value of 0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt**.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **ps**: print the contents of the screen; **pf**: turn off the printer; and **po**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **pO** takes one parameter and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **pf**, is transparently passed to the printer while **pO** is in effect.

Strings to program function keys can be given as **pk**, **pl**, and **px**. Each of these strings takes two parameters: the function key number to program (from 0 to 9) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The differences among the capabilities are that **pk** causes pressing the given key to be the same as the user typing the given string; **pl** causes the string to be executed by the terminal in local mode; and **px** causes the string to be transmitted to the computer. Unfortunately, due to lack of a definition for string parameters in **`termcap`**, only `terminfo`(5) supports these capabilities. The **ZZ** capability is automatically generated in the tgetent call and must never be defined in **`termcap`** entries.

Glitches and Braindamage

Hazeltine terminals, which do not allow '~' characters to be displayed, should indicate **hz**.

The **nc** capability, now obsolete, formerly indicated Datamedia terminals, which echo **\r \n** for carriage return then ignore a following linefeed.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept, should indicate **xn**.

If **ce** is required to get rid of standout (instead of merely writing normal text on top of it), **xs** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a magic cookie, and that to erase standout mode it is necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the ESC or ^C characters, has **xb**, indicating that the "f1" key is used for ESC and "f2" for ^C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form **x** *x*.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last*. The capabilities given before **tc** override those in the terminal type invoked by **tc**. A capability can be canceled by placing **xx@** to the left of the **tc** invocation, where **xx** is the capability. For example, the entry

```
hn|2621-nl:ks@:ke@:tc=2621:
```

defines a "2621-nl" that does not have the **ks** or **ke** capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/usr/share/misc/termcap	File containing terminal descriptions.
/usr/share/misc/termcap.db	Hash database file containing terminal descriptions (see cap_mkdb(1)).

SEE ALSO

cap_mkdb(1), ex(1), more(1), tset(1), ul(1), vi(1), curses(3), printf(3), termcap(3)

HISTORY

The **termcap** file format appeared in 3BSD.

BUGS

Note: The **termcap** functions were replaced by terminfo(5) in AT&T System V UNIX Release 2.0. The transition will be relatively painless if capabilities flagged as "obsolete" are avoided.

Lines and columns are now stored by the kernel as well as in the termcap entry. Most programs now use the kernel information primarily; the information in this file is used only if the kernel does not have any information.

Not all programs support all entries.

NAME

texinfo − software documentation system

DESCRIPTION

Texinfo is a documentation system that uses a single source file to produce both online information and printed output. It is primarily designed for writing software manuals.

For a full description of the Texinfo language and associated tools, please see the Texinfo manual (written in Texinfo itself). Most likely, running this command from your shell:

info texinfo

or this key sequence from inside Emacs:

M-x info RET m texinfo RET

will get you there.

AVAILABILITY

ftp://ftp.gnu.org/gnu/texinfo/
or any GNU mirror site.

REPORTING BUGS

Please send bug reports to bug-texinfo@gnu.org, general questions and discussion to help-texinfo@gnu.org.

SEE ALSO

info(1), install-info(1), makeinfo(1), texi2dvi(1), texindex(1).
emacs(1), tex(1).
info(5).

NAME

transport − Postfix transport table format

SYNOPSIS

postmap /etc/postfix/transport

postmap -q "*string***" /etc/postfix/transport**

postmap -q - /etc/postfix/transport <*inputfile*

DESCRIPTION

The optional **transport**(5) table specifies a mapping from email addresses to message delivery transports and next-hop destinations. Message delivery transports such as **local** or **smtp** are defined in the **master.cf** file, and next-hop destinations are typically hosts or domain names. The table is searched by the **trivial-rewrite**(8) daemon.

This mapping overrides the default *transport*:*nexthop* selection that is built into Postfix:

local_transport (default: local:$myhostname)
: This is the default for final delivery to domains listed with **mydestination**, and for [*ipaddress*] destinations that match **$inet_interfaces** or **$proxy_interfaces**. The default *nexthop* destination is the MTA hostname.

virtual_transport (default: virtual:)
: This is the default for final delivery to domains listed with **virtual_mailbox_domains**. The default *nexthop* destination is the recipient domain.

relay_transport (default: relay:)
: This is the default for remote delivery to domains listed with **relay_domains**. In order of decreasing precedence, the *nexthop* destination is taken from **relay_transport**, **sender_dependent_relayhost_maps**, **relayhost**, or from the recipient domain.

default_transport (default: smtp:)
: This is the default for remote delivery to other destinations. In order of decreasing precedence, the *nexthop* destination is taken from **default_transport**, **sender_dependent_relayhost_maps**, **relayhost**, or from the recipient domain.

Normally, the **transport**(5) table is specified as a text file that serves as input to the **postmap**(1) command. The result, an indexed file in **dbm** or **db** format, is used for fast searching by the mail system. Execute the command "**postmap /etc/postfix/transport**" to rebuild an indexed file after changing the corresponding transport table.

When the table is provided via other means such as NIS, LDAP or SQL, the same lookups are done as for ordinary indexed files.

Alternatively, the table can be provided as a regular-expression map where patterns are given as regular expressions, or lookups can be directed to TCP-based server. In those case, the lookups are done in a slightly different way as described below under "REGULAR EXPRESSION TABLES" or "TCP-BASED TABLES".

CASE FOLDING

The search string is folded to lowercase before database lookup. As of Postfix 2.3, the search string is not case folded with database types such as regexp: or pcre: whose lookup fields can match both upper and lower case.

TABLE FORMAT

The input format for the **postmap**(1) command is as follows:

pattern result
When *pattern* matches the recipient address or domain, use the corresponding *result*.

blank lines and comments
Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a '#'.

multi-line text
A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

The *pattern* specifies an email address, a domain name, or a domain name hierarchy, as described in section "TABLE LOOKUP".

The *result* is of the form *transport:nexthop* and specifies how or where to deliver mail. This is described in section "RESULT FORMAT".

TABLE SEARCH ORDER

With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, patterns are tried in the order as listed below:

user+extension@domain transport:*nexthop*
Deliver mail for *user+extension@domain* through *transport* to *nexthop*.

user@domain transport:*nexthop*
Deliver mail for *user@domain* through *transport* to *nexthop*.

domain transport:*nexthop*
Deliver mail for *domain* through *transport* to *nexthop*.

.*domain transport*:*nexthop*
Deliver mail for any subdomain of *domain* through *transport* to *nexthop*. This applies only when the string **transport_maps** is not listed in the **parent_domain_matches_subdomains** configuration setting. Otherwise, a domain name matches itself and its subdomains.

Note 1: the special pattern ***** represents any address (i.e. it functions as the wild-card pattern).

Note 2: the null recipient address is looked up as **$empty_address_recipient**@**$myhostname** (default: mailer-daemon@hostname).

Note 3: *user@domain* or *user+extension@domain* lookup is available in Postfix 2.0 and later.

RESULT FORMAT

The lookup result is of the form *transport***:***nexthop*. The *transport* field specifies a mail delivery transport such as **smtp** or **local**. The *nexthop* field specifies where and how to deliver mail.

The transport field specifies the name of a mail delivery transport (the first name of a mail delivery service entry in the Postfix **master.cf** file).

The interpretation of the nexthop field is transport dependent. In the case of SMTP, specify a service on a non-default port as *host*:*service*, and disable MX (mail exchanger) DNS lookups with [*host*] or [*host*]:*port*. The [] form is required when you specify an IP address instead of a hostname.

A null *transport* and null *nexthop* result means "do not change": use the delivery transport and nexthop information that would be used when the entire transport table did not exist.

A non-null *transport* field with a null *nexthop* field resets the nexthop information to the recipient domain.

A null *transport* field with non-null *nexthop* field does not modify the transport information.

EXAMPLES

In order to deliver internal mail directly, while using a mail relay for all other mail, specify a null entry for internal destinations (do not change the delivery transport or the nexthop information) and specify a wildcard for all other destinations.

```
my.domain   :
.my.domain  :
*           smtp:outbound-relay.my.domain
```

In order to send mail for **example.com** and its subdomains via the **uucp** transport to the UUCP host named **example**:

```
example.com     uucp:example
.example.com    uucp:example
```

When no nexthop host name is specified, the destination domain name is used instead. For example, the following directs mail for *user*@**example.com** via the **slow** transport to a mail exchanger for **example.com**. The **slow** transport could be configured to run at most one delivery process at a time:

```
example.com     slow:
```

When no transport is specified, Postfix uses the transport that matches the address domain class (see DESCRIPTION above). The following sends all mail for **example.com** and its subdomains to host **gateway.example.com**:

```
example.com     :[gateway.example.com]
.example.com    :[gateway.example.com]
```

In the above example, the [] suppress MX lookups. This prevents mail routing loops when your machine is primary MX host for **example.com**.

In the case of delivery via SMTP, one may specify *hostname*:*service* instead of just a host:

```
example.com     smtp:bar.example:2025
```

This directs mail for *user*@**example.com** to host **bar.example** port **2025**. Instead of a numerical port a symbolic name may be used. Specify [] around the hostname if MX lookups must be disabled.

The error mailer can be used to bounce mail:

```
.example.com    error:mail for *.example.com is not deliverable
```

This causes all mail for *user@anything***.example.com** to be bounced.

REGULAR EXPRESSION TABLES

This section describes how the table lookups change when the table is given in the form of regular expressions. For a description of regular expression lookup table syntax, see **regexp_table**(5) or **pcre_table**(5).

Each pattern is a regular expression that is applied to the entire address being looked up. Thus, *some.domain.hierarchy* is not looked up via its parent domains, nor is *user+foo@domain* looked up as *user@domain*.

Patterns are applied in the order as specified in the table, until a pattern is found that matches the search string.

The **trivial-rewrite**(8) server disallows regular expression substitution of $1 etc. in regular expression lookup tables, because that could open a security hole (Postfix version 2.3 and later).

TCP-BASED TABLES

This section describes how the table lookups change when lookups are directed to a TCP-based server. For a description of the TCP client/server lookup protocol, see **tcp_table**(5). This feature is not available up to and including Postfix version 2.4.

Each lookup operation uses the entire recipient address once. Thus, *some.domain.hierarchy* is not looked up via its parent domains, nor is *user+foo@domain* looked up as *user@domain*.

Results are the same as with indexed file lookups.

CONFIGURATION PARAMETERS

The following **main.cf** parameters are especially relevant. The text below provides only a parameter summary. See **postconf**(5) for more details including examples.

empty_address_recipient
: The address that is looked up instead of the null sender address.

parent_domain_matches_subdomains
: List of Postfix features that use *domain.tld* patterns to match *sub.domain.tld* (as opposed to requiring *.domain.tld* patterns).

transport_maps
: List of transport lookup tables.

SEE ALSO

trivial-rewrite(8), rewrite and resolve addresses
postconf(5), configuration parameters
postmap(1), Postfix lookup table manager

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
ADDRESS_REWRITING_README, address rewriting guide
DATABASE_README, Postfix lookup table overview
FILTER_README, external content filter

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

ttyaction — ttyaction file format

DESCRIPTION

The **ttyaction** file specifies site-specific commands to run when a login session begins and ends. The **ttyaction** file contains a list of newline separated records, where each record has the following three fields:

ttyname — Name of the tty line(s) on which this line should apply. The name is relative to the `/dev` directory, similar to how such devices are named in the `/etc/ttys` file.

action — Name of the action for which this line should apply. The action names currently defined are "login", "getty", "telnetd" and "rlogind" which indicate which program is processing this file. (Note that "login" begins a login session, while the other three are run after a login session ends.)

command — What command to run if this record matches.

The first two fields are delimited with blanks or tabs, and the command field is all text to the end of the line. Either or both of first two fields may contain wildcard match patterns as implemented by the **fnmatch**() library function.

All command strings are executed by passing them to `/bin/sh -c` running as "root," with an environment containing:

```
TTY=ttyname
ACT=action
USER=username
PATH=_PATH_STDPATH
```

These variables may be used directly in the shell command part of the record for simple tasks such as changing the ownership of related devices. For example:

```
console  *      chown ${USER}:tty /dev/mouse
```

will *chown* the mouse appropriately when the console owner changes.

EXAMPLES

Here are some more example records:

```
tty0    login   /somewhere/tty_setup ${TTY}
tty0    getty   /somewhere/tty_clean ${TTY}
*       *       /somewhere/ttyfrob ${TTY} ${ACT}
```

SEE ALSO

`fnmatch`(3), `ttyaction`(3)

HISTORY

The ideas for the `/etc/ttyaction` file were inspired by the `/etc/fbtab` file under SunOS.

NAME

ttys — terminal initialization information

DESCRIPTION

The file **ttys** contains information that is used by various routines to initialize and control the use of terminal special files. This information is read with the getttyent(3) library routines.

There is one line in the **ttys** file per special device file. Fields are separated by tabs and/or spaces. Fields comprising more than one word should be enclosed in double quotes (‘"’). Blank lines and comments may appear anywhere in the file; comments are delimited by hash marks (‘#’) and new lines. Any unspecified fields will default to null.

Each line in **ttys** has the format:

```
tty command type flags
```

The first field is the name of the terminal special file as it is found in /dev.

The second field of the file is the command to execute for the line, usually getty(8), which initializes and opens the line, setting the speed, waiting for a user name and executing the login(1) program. However, it can be any desired command, for example the start up for a window system terminal emulator or some other daemon process, and can contain multiple words if quoted.

The third field is the type of terminal usually connected to that tty line, normally the one found in the termcap(5) data base file. The environment variable TERM is initialized with the value by either getty(8) or login(1).

The remaining fields set flags in the *ty_status* entry (see getttyent(3)) or specify a window system process that init(8) will maintain for the terminal line or a key into a database of tty attributes (currently unused).

on or **off** init(8) should (or should not) execute the command given in the second field.

secure If **on** is specified, allows users with a uid of 0 (e.g. "root") to login on this line.

local Sets the TIOCFLAG_CLOCAL tty(4) flag for the device. This will cause the termios(4) CLOCAL flag to be set on every open and thus modem control signal lines will be ignored by default.

softcar Causes the driver to ignore hardware carrier on the line (by setting the TIOCFLAG_SOFTCAR tty(4) flag).

rtscts Sets the TIOCFLAG_CRTSCTS tty(4) flag for the device to enable RTS / CTS "hardware" flow control by default.

mdmbuf Sets the TIOCFLAG_MDMBUF tty(4) flag for the device to enable DTR / DCD "hardware" flow control by default.

The flags "local", "rtscts", "mdmbuf", and "softcar" modify the default behaviour of the terminal line, and their actions are device driver dependent. These flag fields should not be quoted.

The string ‘‘window=’’ may be followed by a quoted command string which init(8) will execute *before* starting the command specified by the second field.

The string ‘‘class=’’ may be followed by a quoted string used as a key into a database of attributes for that category of tty. See getttynam(3) for more information on this feature.

After changing the **ttys** file a SIGHUP signal can be sent to init(8) with the command “kill -s HUP 1”. On receipt of this signal, init(8) will re-read the **ttys** file and spawn any necessary getty(8) processes.

Nota Bene: Sending `SIGHUP` to `init`(8) does *not* change the state of the various `tty`(4) device flags listed above; the `ttyflags`(8) program must be run for changes in those flags to take effect on the devices.

FILES

`/etc/ttys`

EXAMPLES

```
# root login on console at 1200 baud
console "/usr/libexec/getty std.1200" vt100   on secure
# dialup at 1200 baud, no root logins
ttyd0   "/usr/libexec/getty d1200"    dialup  on      # 555-1234
# Mike's terminal: hp2621
ttyh0   "/usr/libexec/getty std.9600" hp2621-nl      on      # 457 Evans
# John's terminal: vt100
ttyh1   "/usr/libexec/getty std.9600" vt100   on              # 459 Evans
# terminal emulate/window system
ttyv0   "/usr/new/xterm -L :0"        vs100   on window="/usr/new/Xvs100 0"
# Network pseudo ttys -- don't enable getty
ttyp0   none    network
ttyp1   none    network off
```

SEE ALSO

`login`(1), `getttyent`(3), `ttyslot`(3), `tty`(4), `gettytab`(5), `termcap`(5), `getty`(8), `init`(8), `ttyflags`(8)

HISTORY

A **`ttys`** file appeared in Version 6 AT&T UNIX.

NAME

types — system data types

SYNOPSIS

```
#include <sys/types.h>
```

DESCRIPTION

The file ⟨sys/types.h⟩ contains the defined data types used in the kernel (most are used throughout the system).

Note that in NetBSD code, when an integer of a specific width is needed, the C99 style *int8_t, uint8_t, int16_t, uint16_t, int32_t, uint32_t, int64_t, uint64_t* should be used. The old BSD style *u_int8_t, u_int16_t, u_int32_t, u_int64_t* are deprecated.

```
#ifndef _SYS_TYPES_H_
#define _SYS_TYPES_H_

/* Machine type dependent parameters. */
#include <machine/types.h>

#include <machine/ansi.h>
#include <machine/endian.h>

#if !defined(_POSIX_SOURCE) && !defined(_XOPEN_SOURCE)
typedef unsigned char  u_char;
typedef unsigned short u_short;
typedef unsigned int   u_int;
typedef unsigned long  u_long;

typedef unsigned char  unchar;        /* Sys V compatibility */
typedef unsigned short ushort;        /* Sys V compatibility */
typedef unsigned int   uint;          /* Sys V compatibility */
typedef unsigned long  ulong;         /* Sys V compatibility */
#endif

typedef uint64_t       u_quad_t;      /* quads */
typedef int64_t        quad_t;
typedef quad_t *       qaddr_t;

typedef char *         caddr_t;       /* core address */
typedef int32_t        daddr_t;       /* disk address */
typedef int32_t        dev_t;         /* device number */
typedef uint32_t       fixpt_t;       /* fixed point number */
typedef uint32_t       gid_t;         /* group id */
typedef uint32_t       ino_t;         /* inode number */
typedef long           key_t;         /* IPC key (for Sys V IPC) */
typedef uint16_t       mode_t;        /* permissions */
typedef uint16_t       nlink_t;       /* link count */
typedef quad_t         off_t;         /* file offset */
typedef int32_t        pid_t;         /* process id */
typedef quad_t         rlim_t;        /* resource limit */
typedef int32_t        segsz_t;       /* segment size */
typedef int32_t        swblk_t;       /* swap offset */
```

```
typedef uint32_t       uid_t;          /* user id */

#if !defined(_POSIX_SOURCE) && !defined(_XOPEN_SOURCE)
/* Major, minor numbers, dev_t's. */
#define major(x)       ((int32_t)(((uint32_t)(x) >> 8) & 0xff))
#define minor(x)       ((int32_t)((x) & 0xff))
#define makedev(x,y)   ((dev_t)(((x) << 8) | (y)))
#endif

#ifdef  _BSD_CLOCK_T_
typedef _BSD_CLOCK_T_  clock_t;
#undef  _BSD_CLOCK_T_
#endif

#ifdef  _BSD_SIZE_T_
typedef _BSD_SIZE_T_   size_t;
#undef  _BSD_SIZE_T_
#endif

#ifdef  _BSD_SSIZE_T_
typedef _BSD_SSIZE_T_  ssize_t;
#undef  _BSD_SSIZE_T_
#endif

#ifdef  _BSD_TIME_T_
typedef _BSD_TIME_T_   time_t;
#undef  _BSD_TIME_T_
#endif

#ifdef  _BSD_CLOCKID_T_
typedef _BSD_CLOCKID_T_        clockid_t;
#undef  _BSD_CLOCKID_T_
#endif

#ifdef  _BSD_TIMER_T_
typedef _BSD_TIMER_T_  timer_t;
#undef  _BSD_TIMER_T_
#endif

#if !defined(_POSIX_SOURCE) && !defined(_XOPEN_SOURCE)
#define NBBY    8              /* number of bits in a byte */

/*
 * Select uses bit masks of file descriptors in longs.  These macros
 * manipulate such bit fields (the filesystem macros use chars).
 * FD_SETSIZE may be defined by the user, but the default here should
 * be enough for most uses.
 */
#ifndef FD_SETSIZE
#define FD_SETSIZE     256
#endif
```

```
typedef int32_t       fd_mask;
#define NFDBITS       (sizeof(fd_mask) * NBBY)      /* bits per mask */

#ifndef howmany
#define howmany(x, y)  (((x) + ((y) - 1)) / (y))
#endif

typedef struct fd_set {
        fd_mask fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;

#define FD_SET(n, p)   ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n, p)   ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p) ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_COPY(f, t)  memcpy(t, f, sizeof(*(f)))
#define FD_ZERO(p)     memset(p, 0, sizeof(*(p)))

#endif /* !defined(_POSIX_SOURCE) ... */
#endif /* !_SYS_TYPES_H_ */
```

SEE ALSO

lseek(2), select(2), truncate(2), byteorder(3), time(3), fs(5)

HISTORY

A **types** file appeared in Version 7 AT&T UNIX.

NAME

tzfile — time zone information

SYNOPSIS

```
#include <tzfile.h>
```

DESCRIPTION

The time zone information files used by tzset(3) begin with the magic characters "TZif" to identify them as time zone information files, followed by sixteen bytes reserved for future use, followed by six four-byte values of type *long*, written in a "standard" byte order (the high-order byte of the value is written first). These values are, in order:

tzh_ttisgmtcnt
: The number of UTC/local indicators stored in the file.

tzh_ttisstdcnt
: The number of standard/wall indicators stored in the file.

tzh_leapcnt
: The number of leap seconds for which data is stored in the file.

tzh_timecnt
: The number of "transition times" for which data is stored in the file.

tzh_typecnt
: The number of "local time types" for which data is stored in the file (must not be zero).

tzh_charcnt
: The number of characters of "time zone abbreviation strings" stored in the file.

The above header is followed by *tzh_timecnt* four-byte values of type *long*, sorted in ascending order. These values are written in "standard" byte order. Each is used as a transition time (as returned by time(3)) at which the rules for computing local time change. Next come *tzh_timecnt* one-byte values of type *unsigned char*; each one tells which of the different types of "local time" types described in the file is associated with the same-indexed transition time. These values serve as indices into an array of *ttinfo* structures that appears next in the file; these structures are defined as follows:

```
struct ttinfo {
        long    tt_gmtoff;
        int     tt_isdst;
        unsigned int   tt_abbrind;
};
```

Each structure is written as a four-byte value for *tt_gmtoff* of type *long*, in a standard byte order, followed by a one-byte value for *tt_isdst* and a one-byte value for *tt_abbrind*. In each structure, *tt_gmtoff* gives the number of seconds to be added to UTC, *tt_isdst* tells whether *tm_isdst* should be set by localtime(3) and *tt_abbrind* serves as an index into the array of time zone abbreviation characters that follow the *ttinfo* structure(s) in the file.

Then there are *tzh_leapcnt* pairs of four-byte values, written in standard byte order; the first value of each pair gives the time (as returned by time(3)) at which a leap second occurs; the second gives the *total* number of leap seconds to be applied after the given time. The pairs of values are sorted in ascending order by time.

Then there are *tzh_ttisstdcnt* standard/wall indicators, each stored as a one-byte value; they tell whether the transition times associated with local time types were specified as standard time or wall clock time, and are used when a time zone file is used in handling POSIX-style time zone environment variables.

Finally there are *tzh_ttisgmtcnt* UTC/local indicators, each stored as a one-byte value; they tell whether the transition times associated with local time types were specified as UTC or local time, and are used when a time zone file is used in handling POSIX-style time zone environment variables.

`localtime`(3) uses the first standard-time *`ttinfo`* structure in the file (or simply the first *`ttinfo`* structure in the absence of a standard-time structure) if either *tzh_timecnt* is zero or the time argument is less than the first transition time recorded in the file.

SEE ALSO

`ctime`(3), `localtime`(3), `time`(3)

NAME

usermgmt.conf — user management tools configuration file

SYNOPSIS

usermgmt.conf

DESCRIPTION

The **usermgmt.conf** file defines the default values used by the user management tools, useradd(8) and friends.

Options in this file can be set by manually editing /etc/usermgmt.conf or using the **-D** option to useradd(8).

base_dir sets the base directory name, in which new users' home directories are created when using the **-m** option to useradd(8).

class sets the default login class for new users. See login.conf(5) for more information on user login classes.

expire sets the default time at which the current password expires. This can be used to implement password aging. Both the *expire* and *inactive* fields should be entered in the form "month day year", where month is the month name (the first three characters are sufficient), day is the day of the month, and year is the year. Time in seconds since the epoch (UTC) is also valid. A value of 0 can be used to disable this feature.

group sets the default primary group for new users. If this is =uid, then a uid and gid will be picked which are both unique and the same, and a line added to /etc/group to describe the new group. It has the format:

group *gid* | *name* | =uid

homeperm sets the default permissions of the newly created home directory if **-m** is given to useradd(8). The permission is specified as an octal number, with or without a leading zero.

inactive sets the default time at which new accounts expire. A value of 0 can be used to disable this feature. Also see the *expire* field.

password specifies an already-encrypted default password.

preserve If this value is one of true, yes, or a non-zero number, then the user login information will be preserved when removing a user with userdel(8).

range specifies the uid boundaries for new users. If unspecified, the default is "1000..60000". It has the format:

range *starting-uid..ending-uid*

shell sets the default login shell for new users.

skel_dir sets the default skeleton directory in which to find files with which to populate the new user's home directory.

FILES

/etc/usermgmt.conf
/etc/skel/*
/etc/login.conf

SEE ALSO

`login.conf`(5), `passwd`(5), `user`(8), `useradd`(8), `userdel`(8), `usermod`(8)

HISTORY

The **`usermgmt.conf`** configuration file first appeared in NetBSD 1.5.

NAME

utmp, **wtmp**, **lastlog** — login records

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

The file ⟨utmp.h⟩ declares the structures used to record information about current users in the file **utmp**, logins and logouts in the file **wtmp**, and last logins in the file **lastlog**. The time stamps of date changes, shutdowns and reboots are also logged in the **wtmp** file.

The **wtmp** file can grow rapidly on busy systems, and is normally rotated with newsyslog(8).

These files must be created manually; if they do not exist, they are not created automatically.

```
#define _PATH_UTMP      "/var/run/utmp"
#define _PATH_WTMP      "/var/log/wtmp"
#define _PATH_LASTLOG   "/var/log/lastlog"

#define UT_NAMESIZE     8
#define UT_LINESIZE     8
#define UT_HOSTSIZE     16

struct lastlog {
        time_t  ll_time;
        char    ll_line[UT_LINESIZE];
        char    ll_host[UT_HOSTSIZE];
};

struct utmp {
        char    ut_line[UT_LINESIZE];
        char    ut_name[UT_NAMESIZE];
        char    ut_host[UT_HOSTSIZE];
        time_t ut_time;
};
```

Each time a user logs in, the login(1) program looks up the user's UID in the file **lastlog**. If it is found, the timestamp of the last time the user logged in, the terminal line and the hostname are written to the standard output, providing the login is not set *quiet*; see login(1). The login(1) program then records the new login time in the file **lastlog**.

After the new *lastlog* record is written, the file **utmp** is opened and the *utmp* record for the user inserted. This record remains there until the user logs out at which time it is deleted (by clearing the user and host fields, and updating the timestamp field). The **utmp** file is used by the programs rwho(1), users(1), w(1), and who(1).

Next, the login(1) program opens the file **wtmp**, and appends the user's *utmp* record. When the user logs out, a *utmp* record with the tty line, an updated time stamp, and cleared user and host fields is appended to the file by init(8). The **wtmp** file is used by the programs last(1) and ac(8).

In the event of a date change, a shutdown or reboot, the following items are logged in the **wtmp** file.

reboot

`shutdown` A system reboot or shutdown has been initiated. The character '~' is placed in the field *ut_line*, and `reboot` or `shutdown` in the field *ut_name* (see `shutdown`(8) and `reboot`(8)).

`date` The system time has been manually or automatically updated by `date`(1). The command name *date* is recorded in the field *ut_name*. In the field *ut_line*, the character '|' indicates the time prior to the change, and the character '{' indicates the new time.

FILES

`/var/run/utmp` The **utmp** file.
`/var/log/wtmp` The **wtmp** file.
`/var/log/lastlog` The **lastlog** file.

SEE ALSO

`last`(1), `login`(1), `w`(1), `who`(1), `utmpx`(5), `ac`(8), `init`(8), `lastlogin`(8), `newsyslog`(8)

HISTORY

A **utmp** and **wtmp** file format appeared in Version 6 AT&T UNIX. The **lastlog** file format appeared in 3.0BSD.

NAME

utmpx, **wtmpx**, **lastlogx** — user accounting database

SYNOPSIS

```
#include <utmpx.h>
```

DESCRIPTION

In contrast to utmp and wtmp, the extended databases in utmpx and wtmpx reserve more space for logging hostnames, and also information on a process' ID, termination signal and exit status.

The ⟨utmpx.h⟩ header defines the structures and functions for logging user. Currently logged in users are tracked in /var/run/utmpx, a list of all logins and logouts, as well as all shutdowns, reboots and date changes, is kept in /var/log/wtmpx, and the last login of each user is noted in /var/log/lastlogx.

The interface to the **utmpx** file is described in getutxent(3).

The **wtmpx** file can grow rapidly on busy systems, and is normally rotated with newsyslog(8).

In the event of a date change, a shutdown, or a reboot, the following items are logged in the **wtmpx** file:

- **date** The system time has been manually or automatically updated by date(1). The command name *date* is recorded in the field *ut_name*. In the field *ut_line*, the character '|' indicates the time prior to the change, and the character '{' indicates the new time.
- **reboot**
- **shutdown** A system reboot or shutdown has been initiated. The character '~' is placed in the field *ut_line*, and reboot or shutdown in the field *ut_name* (see shutdown(8) and reboot(8)), using logwtmpx(3).

FILES

/var/run/utmpx	The **utmpx** file.
/var/log/wtmpx	The **wtmpx** file.
/var/log/lastlogx	The **lastlogx** file.

SEE ALSO

last(1), login(1), rwho(1), w(1), who(1), endutxent(3), logwtmpx(3), utmp(5), ac(8), init(8), newsyslog(8), reboot(8)

NAME

uuencode — format of an encoded uuencode file

DESCRIPTION

Files output by `uuencode`(1) consist of a header line, followed by a number of body lines, and a trailer line. The `uudecode`(1) command will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line starts with the word "begin", a space, a file mode (in octal), a space, and finally a string which names the file being encoded.

The central engine of `uuencode`(1) is a six-bit encoding function which outputs an ASCII character. The six bits to be encoded are treated as a small integer and added with the ASCII value for the space character (octal 40). The result is a printable ASCII character. In the case where all six bits to be encoded are zero, the ASCII character ` (octal 140) is emitted instead of what would normally be a space.

The body of an encoded file consists of one or more lines, each of which may be a maximum of 86 characters long (including the trailing newline). Each line represents an encoded chunk of data from the input file and begins with a byte count, followed by encoded bytes, followed by a newline.

The byte count is a six-bit integer encoded with the above function, representing the number of bytes encoded in the rest of the line. The method used to encode the data expands its size by 133% (described below). Therefore it is important to note that the byte count describes the size of the chunk of data before it is encoded, not afterwards. The six bit size of this number effectively limits the number of bytes that can be encoded in each line to a maximum of 63. While `uuencode`(1) will not encode more than 45 bytes per line, `uudecode`(1) will tolerate the maximum line size.

The remaining characters in the line represent the data of the input file encoded as follows. Input data are broken into groups of three eight-bit bytes, which are then interpreted together as a 24-bit block. The first bit of the block is the highest order bit of the first character, and the last is the lowest order bit of the third character. This block is then broken into four six-bit integers which are encoded one by one starting from the first bit of the block. The result is a four character ASCII string for every three bytes of input data.

Encoded lines of data continue in this manner until the input file is exhausted. The end of the body is signaled by an encoded line with a byte count of zero (the ASCII character `).

Obviously, not every input file will be a multiple of three bytes in size. In these cases, `uuencode`(1) will pad the remaining one or two bytes of data with garbage bytes until a three byte group is created. The byte count in a line containing garbage padding will reflect the actual number of bytes encoded, making it possible to convey how many bytes are garbage.

The trailer line consists of "end" on a line by itself.

SEE ALSO

`mail`(1), `uucp`(1), `uudecode`(1), `uuencode`(1), `ascii`(7)

HISTORY

The **uuencode** file format appeared in 4.0BSD.

BUGS

The interpretation of the **uuencode** format relies on properties of the ASCII character set and may not work correctly on non-ASCII systems.

NAME

veriexec — format for the *Veriexec* signatures file

DESCRIPTION

Veriexec loads entries to the in-kernel database from a file describing files to be monitored and the type of monitoring. This file is often referred to as the ‘signatures database’ or ‘signatures file’.

The signatures file can be easily created using veriexecgen(8).

SIGNATURES DATABASE FORMAT

The signatures database has a line based structure, where each line has several fields separated by white-space (space, tabs, etc.) taking the following form:

```
path  type  fingerprint flags
```

The description for each field is as follows:

path The full path to the file. White-space characters can be escaped if prefixed with a ‘\’.

type Type of fingerprinting algorithm used for the file.

Requires kernel support for the specified algorithm. List of fingerprinting algorithms supported by the kernel can be obtained by using the following command:

```
# sysctl kern.veriexec.algorithms
```

fingerprint
The fingerprint for the file. Can (usually) be generated using the following command:

```
% cksum -a <algorithm> <file>
```

flags Optional listing of entry flags, separated by a comma. These may include:

direct Allow direct execution only.

Execution of a program is said to be “direct” when the program is invoked by the user (either in a script, manually typing it, etc.) via the execve(2) syscall.

indirect
Allow indirect execution only.

Execution of a program is said to be “indirect” if it is invoked by the kernel to interpret a script (“hash-bang”).

file Allow opening the file only, via the open(2) syscall (no execution is allowed).

untrusted
Indicate that the file is located on untrusted storage and its fingerprint evaluation status should not be cached, but rather re-calculated each time it is accessed.

Fingerprints for untrusted files will always be evaluated on load.

To improve readaibility of the signatures file, the following aliases are provided:

program
An alias for “direct”.

interpreter
An alias for “indirect”

script An alias for both “direct” and “file”.

library
An alias for both “file” and “indirect”.

If no flags are specified, “direct” is assumed.

Comments begin with a ‘#’ character and span to the end of the line.

SEE ALSO

`veriexec`(4), `security`(8), `veriexec`(8), `veriexecctl`(8), `veriexecgen`(8)

HISTORY

`veriexec` first appeared in NetBSD 2.0.

AUTHORS

Brett Lymn ⟨blymn@NetBSD.org⟩
Elad Efrat ⟨elad@NetBSD.org⟩

NAME

vgrindefs — language definition data base for vgrind(1)

SYNOPSIS

vgrindefs

DESCRIPTION

The **vgrindefs** file contains all language definitions for vgrind(1). The data base is very similar to termcap(5).

FIELDS

The following table names and describes each field.

Name	Type	Description
pb	str	regular expression for start of a procedure
bb	str	regular expression for start of a lexical block
be	str	regular expression for the end of a lexical block
cb	str	regular expression for the start of a comment
ce	str	regular expression for the end of a comment
sb	str	regular expression for the start of a string
se	str	regular expression for the end of a string
lb	str	regular expression for the start of a character constant
le	str	regular expression for the end of a character constant
tl	bool	present means procedures are only defined at the top lexical level
oc	bool	present means upper and lower case are equivalent
kw	str	a list of keywords separated by spaces

EXAMPLES

The following entry, which describes the C language, is typical of a language entry.

```
C|c::pb=^\d?*?\d?\p\d?\(\a?\):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
:lb=':le=\e':tl:\
:kw=asm auto break case char continue default do double else enum\
extern float for fortran goto if int long register return short\
sizeof static struct switch typedef union unsigned while #define\
#else #endif #if #ifdef #ifndef #include #undef # define else endif\
if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to vgrind(1) as "c" or "C".

Entries may continue onto multiple lines by giving a \ as the last character of a line. Capabilities in **vgrindefs** are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list.

REGULAR EXPRESSIONS

vgrindefs uses regular expression which are very similar to those of ex(1) and lex(1). The characters '^', '$', ':' and '\' are reserved characters and must be "quoted" with a preceding '\' if they are to be included as normal characters. The metasymbols and their meanings are:

$ the end of a line

^ the beginning of a line

\d a delimiter (space, tab, newline, start of line)

\a matches any string of symbols (like .* in lex)

\p matches any alphanumeric name. In a procedure definition (pb) the string that matches this symbol is used as the procedure name.

() grouping

| alternation

? last item is optional

\e preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) which can include the string delimiter in a string by escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like "(tramp|steamer)flies?" would match "tramp", "steamer", "trampflies", or "steamerflies".

KEYWORD LIST

The keyword list is just a list of keywords in the language separated by spaces. If the "oc" boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

FILES

`/usr/share/misc/vgrindefs` File containing terminal descriptions.

SEE ALSO

`troff`(1), `vgrind`(1)

HISTORY

The **`vgrindefs`** file format appeared in 4.2BSD.

NAME

virtual − Postfix virtual alias table format

SYNOPSIS

postmap /etc/postfix/virtual

postmap -q "*string***" /etc/postfix/virtual**

postmap -q - /etc/postfix/virtual <*inputfile*

DESCRIPTION

The optional **virtual**(5) alias table rewrites recipient addresses for all local, all virtual, and all remote mail destinations. This is unlike the **aliases**(5) table which is used only for **local**(8) delivery. Virtual aliasing is recursive, and is implemented by the Postfix **cleanup**(8) daemon before mail is queued.

The main applications of virtual aliasing are:

- To redirect mail for one address to one or more addresses.
- To implement virtual alias domains where all addresses are aliased to addresses in other domains.

  Virtual alias domains are not to be confused with the virtual mailbox domains that are implemented with the Postfix **virtual**(8) mail delivery agent. With virtual mailbox domains, each recipient address can have its own mailbox.

Virtual aliasing is applied only to recipient envelope addresses, and does not affect message headers. Use **canonical**(5) mapping to rewrite header and envelope addresses in general.

Normally, the **virtual**(5) alias table is specified as a text file that serves as input to the **postmap**(1) command. The result, an indexed file in **dbm** or **db** format, is used for fast searching by the mail system. Execute the command "**postmap /etc/postfix/virtual**" to rebuild an indexed file after changing the corresponding text file.

When the table is provided via other means such as NIS, LDAP or SQL, the same lookups are done as for ordinary indexed files.

Alternatively, the table can be provided as a regular-expression map where patterns are given as regular expressions, or lookups can be directed to TCP-based server. In those case, the lookups are done in a slightly different way as described below under "REGULAR EXPRESSION TABLES" or "TCP-BASED TABLES".

CASE FOLDING

The search string is folded to lowercase before database lookup. As of Postfix 2.3, the search string is not case folded with database types such as regexp: or pcre: whose lookup fields can match both upper and lower case.

TABLE FORMAT

The input format for the **postmap**(1) command is as follows:

pattern result
: When *pattern* matches a mail address, replace it by the corresponding *result*.

blank lines and comments
: Empty lines and whitespace-only lines are ignored, as are lines whose first non-whitespace character is a ‘#’.

multi-line text
: A logical line starts with non-whitespace text. A line that starts with whitespace continues a logical line.

TABLE SEARCH ORDER

With lookups from indexed files such as DB or DBM, or from networked tables such as NIS, LDAP or SQL, patterns are tried in the order as listed below:

user@domain address, address, ...
Redirect mail for *user@domain* to *address*. This form has the highest precedence.

user address, address, ...
Redirect mail for *user@site* to *address* when *site* is equal to $**myorigin**, when *site* is listed in $**mydestination**, or when it is listed in $**inet_interfaces** or $**proxy_interfaces**.

This functionality overlaps with functionality of the local *aliases*(5) database. The difference is that **virtual**(5) mapping can be applied to non-local addresses.

@domain address, address, ...
Redirect mail for other users in *domain* to *address*. This form has the lowest precedence.

Note: *@domain* is a wild-card. With this form, the Postfix SMTP server accepts mail for any recipient in *domain*, regardless of whether that recipient exists. This may turn your mail system into a backscatter source: Postfix first accepts mail for non-existent recipients and then tries to return that mail as "undeliverable" to the often forged sender address.

RESULT ADDRESS REWRITING

The lookup result is subject to address rewriting:

- When the result has the form *@otherdomain*, the result becomes the same *user* in *otherdomain*. This works only for the first address in a multi-address lookup result.
- When "**append_at_myorigin=yes**", append "**@$myorigin**" to addresses without "@domain".
- When "**append_dot_mydomain=yes**", append "**.$mydomain**" to addresses without ".domain".

ADDRESS EXTENSION

When a mail address localpart contains the optional recipient delimiter (e.g., *user+foo@domain*), the lookup order becomes: *user+foo@domain*, *user@domain*, *user+foo*, *user*, and *@domain*.

The **propagate_unmatched_extensions** parameter controls whether an unmatched address extension (*+foo*) is propagated to the result of table lookup.

VIRTUAL ALIAS DOMAINS

Besides virtual aliases, the virtual alias table can also be used to implement virtual alias domains. With a virtual alias domain, all recipient addresses are aliased to addresses in other domains.

Virtual alias domains are not to be confused with the virtual mailbox domains that are implemented with the Postfix **virtual**(8) mail delivery agent. With virtual mailbox domains, each recipient address can have its own mailbox.

With a virtual alias domain, the virtual domain has its own user name space. Local (i.e. non-virtual) usernames are not visible in a virtual alias domain. In particular, local **aliases**(5) and local mailing lists are not visible as *localname@virtual-alias.domain*.

Support for a virtual alias domain looks like:

/etc/postfix/main.cf:
```
virtual_alias_maps = hash:/etc/postfix/virtual
```

Note: some systems use **dbm** databases instead of **hash**. See the output from "**postconf -m**" for available database types.

/etc/postfix/virtual:

virtual-alias.domain *anything* (right-hand content does not matter)
postmaster@virtual-alias.domain *postmaster*
user1@virtual-alias.domain *address1*
user2@virtual-alias.domain *address2, address3*

The *virtual-alias.domain anything* entry is required for a virtual alias domain. **Without this entry, mail is rejected with "relay access denied", or bounces with "mail loops back to myself".**

Do not specify virtual alias domain names in the **main.cf mydestination** or **relay_domains** configuration parameters.

With a virtual alias domain, the Postfix SMTP server accepts mail for *known-user@virtual-alias.domain*, and rejects mail for *unknown-user@virtual-alias.domain* as undeliverable.

Instead of specifying the virtual alias domain name via the **virtual_alias_maps** table, you may also specify it via the **main.cf virtual_alias_domains** configuration parameter. This latter parameter uses the same syntax as the **main.cf mydestination** configuration parameter.

REGULAR EXPRESSION TABLES

This section describes how the table lookups change when the table is given in the form of regular expressions. For a description of regular expression lookup table syntax, see **regexp_table**(5) or **pcre_table**(5).

Each pattern is a regular expression that is applied to the entire address being looked up. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Patterns are applied in the order as specified in the table, until a pattern is found that matches the search string.

Results are the same as with indexed file lookups, with the additional feature that parenthesized substrings from the pattern can be interpolated as **$1**, **$2** and so on.

TCP-BASED TABLES

This section describes how the table lookups change when lookups are directed to a TCP-based server. For a description of the TCP client/server lookup protocol, see **tcp_table**(5). This feature is not available up to and including Postfix version 2.4.

Each lookup operation uses the entire address once. Thus, *user@domain* mail addresses are not broken up into their *user* and *@domain* constituent parts, nor is *user+foo* broken up into *user* and *foo*.

Results are the same as with indexed file lookups.

BUGS

The table format does not understand quoting conventions.

CONFIGURATION PARAMETERS

The following **main.cf** parameters are especially relevant to this topic. See the Postfix **main.cf** file for syntax details and for default values. Use the "**postfix reload**" command after a configuration change.

virtual_alias_maps
List of virtual aliasing tables.

virtual_alias_domains
List of virtual alias domains. This uses the same syntax as the **mydestination** parameter.

propagate_unmatched_extensions
A list of address rewriting or forwarding mechanisms that propagate an address extension from the original address to the result. Specify zero or more of **canonical**, **virtual**, **alias**, **forward**, **include**, or **generic**.

Other parameters of interest:

inet_interfaces
The network interface addresses that this system receives mail on. You need to stop and start Postfix when this parameter changes.

mydestination
List of domains that this mail system considers local.

myorigin
The domain that is appended to any address that does not have a domain.

owner_request_special
Give special treatment to **owner-***xxx* and *xxx*-**request** addresses.

proxy_interfaces
Other interfaces that this machine receives mail on by way of a proxy agent or network address translator.

SEE ALSO

cleanup(8), canonicalize and enqueue mail
postmap(1), Postfix lookup table manager
postconf(5), configuration parameters
canonical(5), canonical address mapping

README FILES

Use "**postconf readme_directory**" or "**postconf html_directory**" to locate this information.
ADDRESS_REWRITING_README, address rewriting guide
DATABASE_README, Postfix lookup table overview
VIRTUAL_README, domain hosting guide

LICENSE

The Secure Mailer license must be distributed with this software.

AUTHOR(S)

Wietse Venema
IBM T.J. Watson Research
P.O. Box 704
Yorktown Heights, NY 10598, USA

NAME

weekly.conf — weekly maintenance configuration file

DESCRIPTION

The **weekly.conf** file specifies which of the standard /etc/weekly services are performed. The /etc/weekly script is run, by default, every Saturday morning on a NetBSD system.

The variables described below can be set to "YES" or "NO" in the /etc/weekly.conf file. The default settings are in the /etc/defaults/weekly.conf file. (Note that you should never edit /etc/defaults/weekly.conf directly, as it is often replaced during system upgrades.)

rebuild_locatedb This rebuilds the locate database, /var/db/locate.database, which must also exist, in order to be rebuilt.

rebuild_whatisdb This rebuilds the whatis database(s). Note that NetBSD provides a default whatis.db for the system manual pages and this may not be needed. (Adjust your /etc/man.conf as necessary. See man.conf(5) for details.)

FILES

/etc/weekly	weekly maintenance script
/etc/weekly.conf	weekly maintenance configuration
/etc/weekly.local	local site additions to /etc/weekly

SEE ALSO

daily.conf(5), monthly.conf(5)

HISTORY

The **weekly.conf** file appeared in NetBSD 1.3.

NAME

wpa_supplicant.conf − configuration file for wpa_supplicant

OVERVIEW

wpa_supplicant is configured using a text file that lists all accepted networks and security policies, including pre-shared keys. See the example configuration file, probably in **/usr/share/doc/wpa_supplicant/**, for detailed information about the configuration format and supported fields.

All file paths in this configuration file should use full (absolute, not relative to working directory) path in order to allow working directory to be changed. This can happen if wpa_supplicant is run in the background.

Changes to configuration file can be reloaded be sending SIGHUP signal to **wpa_supplicant** ('killall -HUP wpa_supplicant'). Similarly, reloading can be triggered with the 'wpa_cli reconfigure' command.

Configuration file can include one or more network blocks, e.g., one for each used SSID. wpa_supplicant will automatically select the best network based on the order of network blocks in the configuration file, network security level (WPA/WPA2 is preferred), and signal strength.

QUICK EXAMPLES

1. WPA-Personal (PSK) as home network and WPA-Enterprise with EAP-TLS as work network.

```
# allow frontend (e.g., wpa_cli) to be used by all users in 'wheel' group
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
#
# home network; allow all valid ciphers
network={
        ssid="home"
        scan_ssid=1
        key_mgmt=WPA-PSK
        psk="very secret passphrase"
}
#
# work network; use EAP-TLS with WPA; allow only CCMP and TKIP ciphers
network={
        ssid="work"
        scan_ssid=1
        key_mgmt=WPA-EAP
        pairwise=CCMP TKIP
        group=CCMP TKIP
        eap=TLS
        identity="user@example.com"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
}
```

2. WPA-RADIUS/EAP-PEAP/MSCHAPv2 with RADIUS servers that use old peaplabel (e.g., Funk Odyssey and SBR, Meetinghouse Aegis, Interlink RAD-Series)

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
network={
        ssid="example"
        scan_ssid=1
        key_mgmt=WPA-EAP
```

```
        eap=PEAP
        identity="user@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        phase1="peaplabel=0"
        phase2="auth=MSCHAPV2"
}
```

3. EAP-TTLS/EAP-MD5-Challenge configuration with anonymous identity for the unencrypted use. Real identity is sent only within an encrypted TLS tunnel.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
network={
        ssid="example"
        scan_ssid=1
        key_mgmt=WPA-EAP
        eap=TTLS
        identity="user@example.com"
        anonymous_identity="anonymous@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        phase2="auth=MD5"
}
```

4. IEEE 802.1X (i.e., no WPA) with dynamic WEP keys (require both unicast and broadcast); use EAP-TLS for authentication

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
network={
        ssid="1x-test"
        scan_ssid=1
        key_mgmt=IEEE8021X
        eap=TLS
        identity="user@example.com"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
        eapol_flags=3
}
```

5. Catch all example that allows more or less all configuration modes. The configuration options are used based on what security policy is used in the selected SSID. This is mostly for testing and is not recommended for normal use.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
network={
        ssid="example"
        scan_ssid=1
        key_mgmt=WPA-EAP WPA-PSK IEEE8021X NONE
        pairwise=CCMP TKIP
        group=CCMP TKIP WEP104 WEP40
        psk="very secret passphrase"
```

```
        eap=TTLS PEAP TLS
        identity="user@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
        phase1="peaplabel=0"
        ca_cert2="/etc/cert/ca2.pem"
        client_cert2="/etc/cer/user.pem"
        private_key2="/etc/cer/user.prv"
        private_key2_passwd="password"
}
```

6. Authentication for wired Ethernet. This can be used with

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
ap_scan=0
network={
        key_mgmt=IEEE8021X
        eap=MD5
        identity="user"
        password="password"
        eapol_flags=0
}
```

CERTIFICATES

Some EAP authentication methods require use of certificates. EAP-TLS uses both server side and client certificates whereas EAP-PEAP and EAP-TTLS only require the server side certificate. When client certificate is used, a matching private key file has to also be included in configuration. If the private key uses a passphrase, this has to be configured in wpa_supplicant.conf ("private_key_passwd").

wpa_supplicant supports X.509 certificates in PEM and DER formats. User certificate and private key can be included in the same file.

If the user certificate and private key is received in PKCS#12/PFX format, they need to be converted to suitable PEM/DER format for wpa_supplicant. This can be done, e.g., with following commands:

```
# convert client certificate and private key to PEM format
openssl pkcs12 -in example.pfx -out user.pem -clcerts
# convert CA certificate (if included in PFX file) to PEM format
openssl pkcs12 -in example.pfx -out ca.pem -cacerts -nokeys
```

SEE ALSO

wpa_supplicant(8) **openssl**(1)

NAME

wpa_supplicant.conf — configuration file for wpa_supplicant(8)

DESCRIPTION

The wpa_supplicant(8) utility is an implementation of the WPA Supplicant component, i.e., the part that runs in the client stations. It implements WPA key negotiation with a WPA Authenticator and EAP authentication with Authentication Server using configuration information stored in a text file.

The configuration file consists of optional global parameter settings and one or more network blocks, e.g. one for each used SSID. The wpa_supplicant(8) utility will automatically select the best network based on the order of the network blocks in the configuration file, network security level (WPA/WPA2 is preferred), and signal strength. Comments are indicated with the '#' character; all text to the end of the line will be ignored.

GLOBAL PARAMETERS

Default parameters used by wpa_supplicant(8) may be overridden by specifying

```
parameter=value
```

in the configuration file (note no spaces are allowed). Values with embedded spaces must be enclosed in quote marks.

The following parameters are recognized:

ctrl_interface
: The pathname of the directory in which wpa_supplicant(8) creates UNIX domain socket files for communication with frontend programs such as wpa_cli(8).

ctrl_interface_group
: A group name or group ID to use in setting protection on the control interface file. This can be set to allow non-root users to access the control interface files. If no group is specified, the group ID of the control interface is not modified and will, typically, be the group ID of the directory in which the socket is created.

eapol_version
: The IEEE 802.1x/EAPOL protocol version to use; either 1 (default) or 2. The wpa_supplicant(8) utility is implemented according to IEEE 802-1X-REV-d8 which defines EAPOL version to be 2. However, some access points do not work when presented with this version so by default wpa_supplicant(8) will announce that it is using EAPOL version 1. If version 2 must be announced for correct operation with an access point, this value may be set to 2.

ap_scan Access point scanning and selection control; one of 0, 1 (default), or 2. Only setting 1 should be used with the wlan(4) module; the other settings are for use on other operating systems.

fast_reauth
: EAP fast re-authentication; either 1 (default) or 0. Control fast re-authentication support in EAP methods that support it.

NETWORK BLOCKS

Each potential network/access point should have a "network block" that describes how to identify it and how to set up security. When multiple network blocks are listed in a configuration file, the highest priority one is selected for use or, if multiple networks with the same priority are identified, the first one listed in the configuration file is used.

A network block description is of the form:

```
network={
        parameter=value
        ...
}
```

(note the leading "`network={`" may have no spaces). The block specification contains one or more parameters from the following list:

ssid (required)
: Network name (as announced by the access point). An ASCII or hex string enclosed in quotation marks.

scan_ssid
: SSID scan technique; 0 (default) or 1. Technique 0 scans for the SSID using a broadcast Probe Request frame while 1 uses a directed Probe Request frame. Access points that cloak themselves by not broadcasting their SSID require technique 1, but beware that this scheme can cause scanning to take longer to complete.

bssid
: Network BSSID (typically the MAC address of the access point).

priority
: The priority of a network when selecting among multiple networks; a higher value means a network is more desirable. By default networks have priority 0. When multiple networks with the same priority are considered for selection, other information such as security policy and signal strength are used to select one.

mode
: IEEE 802.11 operation mode; either 0 (infrastructure, default) or 1 (IBSS). Note that IBSS (adhoc) mode can only be used with *key_mgmt* set to `NONE` (plaintext and static WEP).

proto
: List of acceptable protocols; one or more of: `WPA` (IEEE 802.11i/D3.0) and `RSN` (IEEE 802.11i). `WPA2` is another name for `RSN`. If not set this defaults to "`WPA RSN`".

key_mgmt
: List of acceptable key management protocols; one or more of: `WPA-PSK` (WPA pre-shared key), `WPA-EAP` (WPA using EAP authentication), `IEEE8021X` (IEEE 802.1x using EAP authentication and, optionally, dynamically generated WEP keys), `NONE` (plaintext or static WEP keys). If not set this defaults to "`WPA-PSK WPA-EAP`".

auth_alg
: List of allowed IEEE 802.11 authentication algorithms; one or more of: `OPEN` (Open System authentication, required for WPA/WPA2), `SHARED` (Shared Key authentication), `LEAP` (LEAP/Network EAP). If not set automatic selection is used (Open System with LEAP enabled if LEAP is allowed as one of the EAP methods).

pairwise
: List of acceptable pairwise (unicast) ciphers for WPA; one or more of: `CCMP` (AES in Counter mode with CBC-MAC, RFC 3610, IEEE 802.11i/D7.0), `TKIP` (Temporal Key Integrity Protocol, IEEE 802.11i/D7.0), `NONE` (deprecated). If not set this defaults to "`CCMP TKIP`".

group
: List of acceptable group (multicast) ciphers for WPA; one or more of: `CCMP` (AES in Counter mode with CBC-MAC, RFC 3610, IEEE 802.11i/D7.0), `TKIP` (Temporal Key Integrity Protocol, IEEE 802.11i/D7.0), `WEP104` (WEP with 104-bit key), `WEP40` (WEP with 40-bit key). If not set this defaults to "`CCMP TKIP WEP104 WEP40`".

psk
: WPA preshared key used in WPA-PSK mode. The key is specified as 64 hex digits or as an 8-63 character ASCII passphrase. ASCII passphrases are converted to a 256-bit key using the network SSID by the `wpa_passphrase`(8) utility.

eapol_flags
: Dynamic WEP key usage for non-WPA mode, specified as a bit field. Bit 0 (1) forces dynamically generated unicast WEP keys to be used. Bit 1 (2) forces dynamically generated broadcast WEP

keys to be used. By default this is set to 3 (use both).

eap List of acceptable EAP methods; one or more of: `MD5` (EAP-MD5, cannot be used with WPA, used only as a Phase 2 method with EAP-PEAP or EAP-TTLS), `MSCHAPV2` (EAP-MSCHAPV2, cannot be used with WPA; used only as a Phase 2 method with EAP-PEAP or EAP-TTLS), `OTP` (EAP-OTP, cannot be used with WPA; used only as a Phase 2 method with EAP-PEAP or EAP-TTLS), `GTC` (EAP-GTC, cannot be used with WPA; used only as a Phase 2 method with EAP-PEAP or EAP-TTLS), `TLS` (EAP-TLS, client and server certificate), `PEAP` (EAP-PEAP, with tunneled EAP authentication), `TTLS` (EAP-TTLS, with tunneled EAP or PAP/CHAP/MSCHAP/MSCHAPV2 authentication). If not set this defaults to all available methods compiled in to `wpa_supplicant`(8). Note that by default `wpa_supplicant`(8) is compiled with EAP support

identity Identity string for EAP.

anonymous_identity
Anonymous identity string for EAP (to be used as the unencrypted identity with EAP types that support different tunneled identities; e.g. EAP-TTLS).

password
Password string for EAP.

ca_cert Pathname to CA certificate file. This file can have one or more trusted CA certificates. If *ca_cert* is not included, server certificates will not be verified (not recommended).

client_cert
Pathname to client certificate file (PEM/DER).

private_key
Pathname to a client private key file (PEM/DER/PFX). When a PKCS#12/PFX file is used, then *client_cert* should not be specified as both the private key and certificate will be read from PKCS#12 file.

private_key_passwd
Password for any private key file.

dh_file Pathname to a file holding DH/DSA parameters (in PEM format). This file holds parameters for an ephemeral DH key exchange. In most cases, the default RSA authentication does not use this configuration. However, it is possible to set up RSA to use an ephemeral DH key exchange. In addition, ciphers with DSA keys always use ephemeral DH keys. This can be used to achieve forward secrecy. If the *dh_file* is in DSA parameters format, it will be automatically converted into DH params.

subject_match
Substring to be matched against the subject of the authentication server certificate. If this string is set, the server certificate is only accepted if it contains this string in the subject. The subject string is in following format:

```
/C=US/ST=CA/L=San Francisco/CN=Test AS/emailAddress=as@example.com
```

phase1 Phase1 (outer authentication, i.e., TLS tunnel) parameters (string with field-value pairs, e.g., "`peapver=0`" or "`peapver=1 peaplabel=1`").

`peapver` can be used to force which PEAP version (0 or 1) is used.

`peaplabel=1` can be used to force new label, "client PEAP encryption", to be used during key derivation when PEAPv1 or newer. Most existing PEAPv1 implementations seem to be using the old label, "`client EAP encryption`", and `wpa_supplicant`(8) is now using that as the

default value. Some servers, e.g., Radiator, may require `peaplabel=1` configuration to interoperate with PEAPv1; see `eap_testing.txt` for more details.

`peap_outer_success=0` can be used to terminate PEAP authentication on tunneled EAP-Success. This is required with some RADIUS servers that implement `draft-josefsson-pppext-eap-tls-eap-05.txt` (e.g., Lucent NavisRadius v4.4.0 with PEAP in "IETF Draft 5" mode).

`include_tls_length=1` can be used to force `wpa_supplicant`(8) to include TLS Message Length field in all TLS messages even if they are not fragmented.

`sim_min_num_chal=3` can be used to configure EAP-SIM to require three challenges (by default, it accepts 2 or 3)

`fast_provisioning=1` option enables in-line provisioning of EAP-FAST credentials (PAC).

phase2 phase2: Phase2 (inner authentication with TLS tunnel) parameters (string with field-value pairs, e.g., "`auth=MSCHAPV2`" for EAP-PEAP or "`autheap=MSCHAPV2 autheap=MD5`" for EAP-TTLS).

ca_cert2 Like *ca_cert* but for EAP inner Phase 2.

client_cert2
Like *client_cert* but for EAP inner Phase 2.

private_key2
Like *private_key* but for EAP inner Phase 2.

private_key2_passwd
Like *private_key_passwd* but for EAP inner Phase 2.

dh_file2 Like *dh_file* but for EAP inner Phase 2.

subject_match2
Like *subject_match* but for EAP inner Phase 2.

eappsk 16-byte pre-shared key in hex format for use with EAP-PSK.

nai User NAI for use with EAP-PSK.

server_nai
Authentication Server NAI for use with EAP-PSK.

pac_file Pathname to the file to use for PAC entries with EAP-FAST. The `wpa_supplicant`(8) utility must be able to create this file and write updates to it when PAC is being provisioned or refreshed.

eap_workaround
Enable/disable EAP workarounds for various interoperability issues with misbehaving authentication servers. By default these workarounds are enabled. String EAP conformance can be configured by setting this to 0.

CERTIFICATES

Some EAP authentication methods require use of certificates. EAP-TLS uses both server- and client-side certificates, whereas EAP-PEAP and EAP-TTLS only require a server-side certificate. When a client certificate is used, a matching private key file must also be included in configuration. If the private key uses a passphrase, this has to be configured in the **`wpa_supplicant.conf`** file as *private_key_passwd*.

The `wpa_supplicant`(8) utility supports X.509 certificates in PEM and DER formats. User certificate and private key can be included in the same file.

If the user certificate and private key is received in PKCS#12/PFX format, they need to be converted to a suitable PEM/DER format for use by `wpa_supplicant`(8). This can be done using the `openssl`(1) program, e.g. with the following commands:

```
# convert client certificate and private key to PEM format
openssl pkcs12 -in example.pfx -out user.pem -clcerts
# convert CA certificate (if included in PFX file) to PEM format
openssl pkcs12 -in example.pfx -out ca.pem -cacerts -nokeys
```

EXAMPLES

WPA-Personal (PSK) as a home network and WPA-Enterprise with EAP-TLS as a work network:

```
# allow frontend (e.g., wpa_cli) to be used by all users in 'wheel' group
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
#
# home network; allow all valid ciphers
network={
        ssid="home"
        scan_ssid=1
        key_mgmt=WPA-PSK
        psk="very secret passphrase"
}
#
# work network; use EAP-TLS with WPA; allow only CCMP and TKIP ciphers
network={
        ssid="work"
        scan_ssid=1
        key_mgmt=WPA-EAP
        pairwise=CCMP TKIP
        group=CCMP TKIP
        eap=TLS
        identity="user@example.com"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
}
```

WPA-RADIUS/EAP-PEAP/MSCHAPv2 with RADIUS servers that use old peaplabel (e.g., Funk Odyssey and SBR, Meetinghouse Aegis, Interlink RAD-Series):

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
network={
        ssid="example"
        scan_ssid=1
        key_mgmt=WPA-EAP
        eap=PEAP
        identity="user@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        phase1="peaplabel=0"
        phase2="auth=MSCHAPV2"
```

```
}
```

EAP-TTLS/EAP-MD5-Challenge configuration with anonymous identity for the unencrypted use. Real identity is sent only within an encrypted TLS tunnel.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
network={
        ssid="example"
        scan_ssid=1
        key_mgmt=WPA-EAP
        eap=TTLS
        identity="user@example.com"
        anonymous_identity="anonymous@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        phase2="auth=MD5"
}
```

Traditional WEP configuration with 104 bit key specified in hexadecimal. Note the WEP key is not quoted.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
network={
        ssid="example"
        scan_ssid=1
        key_mgmt=NONE
        wep_tx_keyidx=0
        wep_key0=42FEEDDEAFBABEDEAFBEEFAA55
}
```

SEE ALSO

wpa_cli(8), wpa_passphrase(8), wpa_supplicant(8)

HISTORY

The **wpa_supplicant.conf** manual page and wpa_supplicant(8) functionality first appeared in NetBSD 4.0.

AUTHORS

This manual page is derived from the README and wpa_supplicant.conf files in the **wpa_supplicant** distribution provided by Jouni Malinen ⟨jkmaline@cc.hut.fi⟩.

NAME

wscons.conf — workstation console config file

SYNOPSIS

wscons.conf

DESCRIPTION

The **wscons.conf** file defines parameters regarding to the workstation console (wscons). The file consists of lines starting with a keyword, and one or more arguments. Empty lines and lines starting with a hash ("#") are ignored.

The following keywords and arguments are recognized:

font *name width height enc file*
: Used to load a font via wsfontload(8). *name* gives a font name that can be used later, *width* can be used to specify the width of a font character in pixel, *height* is the same, just for the font characters' height. *enc* is used to declare the font's encoding, see the description on wsfontload(8)'s **-e** option for more detail. *file* gives the absolute path to the font file. See wsfontload(8) for more information.

screen *idx scr emul*
: Add and configure virtual console number *idx* using a screen type of *scr* (e.g. 80x25) and a *emul* terminal emulation (e.g. vt100). See wsconscfg(8) for further parameter description.

keyboard *kbd*
: Attach and configure keyboard *kbd* using "wsconscfg -k". If *kbd* is '-' or 'auto', the first free keyboard will be used. See wsconscfg(8) for more information.

encoding *enc*
: Set the keyboard map to the given language code *enc*, using "wsconsctl -w encoding=enc". The map must be supported by the keyboard driver in use and must be compiled into the kernel. See the keyboard driver's manpage (e.g., pckbd(4), ukbd(4)) for details.

mapfile *file*
: Parses the contents of *file*, which contains a keyboard map per line, and calls "wsconsctl -w map+=" for each line. See wsconsctl(8) for details.

mux *idx*
: Used to attach and configure keyboard/mouse multiplexors, using "wsconscfg -m idx". See wsconscfg(8) for more information.

setvar *dev var val*
: Set arbitrary wscons variable *var* to value *val* for specified control device *dev*. Can be used for direct modification of wscons(4) variables, when no other keywords are suitable. See wsconsctl(8) for more information.

Command arguments can be specified as "-" which makes default values come into effect as described in the documentation of the utilities.

FILES

/etc/wscons.conf

SEE ALSO

wscons(4), wsconscfg(8), wsfontload(8)

NAME

wsmoused.conf — multipurpose mouse daemon configuration

SYNOPSIS

wsmoused.conf

DESCRIPTION

The **wsmoused.conf** file configures all the features provided by the wsmoused(8) daemon. It is composed by a series of *blocks*, each of which defines a group of *properties*. The file format is free-form: new line markers are ignored as well as indentation. Comments start with the '#' sign and extend until the end of line.

A *property* is like a variable assignment. It has a name, which goes to the left of the equal sign, and a value, which goes to the right. The assignment ends with a semicolon. It looks like:

```
name = value;
```

There is no difference between string or integer values when defining them. The value must be surrounded by double quotes if it contains whitespace. Booleans are specified as integers, where '0' means false and '1' stands for true. Even though, the program cares about this and will emit a warning if you have done an incorrect assignment. Note that it will not accept unrecognized names.

A *mode* is a type of block that defines how the program behaves when run in a specific mode. A mode inherits properties defined in the global scope. It has the following syntax:

```
mode mode_name {
        property1 = value1;
        ...
        propertyN = valueN;
}
```

There are two recognized modes, action and selection. wsmoused(8) describes what they do in detail.

Properties common to all modes

The following properties can be defined in the global scope, thus affecting all modes, or inside the mode definition, to override global values.

device = pathname;
: The wsmouse(4) device name to use. Defaults to /dev/wsmouse.

fifo = pathname;
: Specify an optional fifo where to redirect all mouse events, no matter if they have been processed. By default, no fifo is used.

modes = string;
: Whitespace separated list of modes to be activated when running. Defaults to 'selection'.

nodaemon = boolean;
: Set to 1 to not fork in the background.

pidfile = basename;
: The basename of the pidfile used to control the process. Pidfiles are always created under /var/run, and have the '.pid' extension automatically added. By default it is set to daemon's program name.

ttystat = pathname;
: `wsdisplay`(4)'s notification device. Defaults to `/dev/ttyEstat`. You will not want to change this.

xconsole = integer;
: Virtual console number which holds the X server (if any). The argument specifies the console number (the same found in `/dev/ttyE?`). Unset by default.

xconsole_delay = integer;
: Number of seconds to wait before reactivating the mouse when returning from the X console (specified by the 'xconsole' property). Defaults to 5.

Properties specific to the action mode

The following properties are only useful when running in the *action* mode:

button_<number>_<status> = command;
: Assigns a command to a button, which will be executed using the `system`(3) call. The 'number' part selects a button to which the command is assigned; the first button is numbered '0' and the maximum depends on the mouse type. The 'status' part can be either 'down' or 'up', representing the events emitted when the button is pressed and released, respectively.

Properties specific to the selection mode

The following properties are only useful when running in the *selection* mode:

lefthanded = boolean;
: Set to 1 to swap mouse buttons, specially useful for left handed users.

slowdown_x = integer;
: X axis slowdown. This positive integer specifies how many events in the vertical direction should be ignored before changing the current column. It defaults to 0.

slowdown_y = integer;
: Y axis slowdown. This positive integer specifies how many events in the horizontal direction should be ignored before changing the current row. It defaults to 3.

FILES

`/etc/wsmoused.conf`	Default configuration file.
`/usr/share/examples/wsmoused/`	Location of sample files.

SEE ALSO

`system`(3), `wsdisplay`(4), `wsmouse`(4), `wsmoused`(8)

HISTORY

The **`wsmoused.conf`** configuration file first appeared in NetBSD 2.0.