## NAME

dictd - a dictionary database server

## SYNOPSIS

**dictd** *[options]*

## DESCRIPTION

**dictd** is a server for the Dictionary Server Protocol (DICT), a TCP transaction based query/response protocol that allows a client to access dictionary definitions from a set of natural language dictionary databases. For security reasons, dictd drops root permissions after startup. If user **dictd** exists on the system, the daemon will run as that user, group **dictd**, otherwise it will run as user **nobody**, group **nobody** or **nogroup** (depending on the operating system distribution). Since startup time is significant, the server is designed to run continuously, and should *not* be run from **inetd**(8). Databases are distributed separately from the server. **dictd** assumes that the index files are sorted alphabeticaly. By default, only alphanumeric charactares are used for search. This default may be overridden by a header in the data file. The only such features implemented at this time are the headers "00-database-allchars" which tells **dictd** that non-alphanumeric characters may also be used for search, and the header "00-database-utf8" which indicates that the database uses utf8 encoding. All headwords in the index file are sorted alphabetically. A header "00-database-plugin" may also be present and is used for integrating plugins into dictd. See "dictfmt_plugin --help" and "dictdplugin.h" for more information. A header "00-database-virtual" identifies "virtual dictionaries", which are lists of real dictionaries to be searched by **dictd.**

## BACKGROUND

For many years, the Internet community has relied on the "webster" protocol for access to natural language definitions. The webster protocol supports access to a single dictionary and (optionally) to a single thesaurus. In recent years, the number of publicly available webster servers on the Internet has dramatically decreased. Fortunately, several freely-distributable dictionaries and lexicons have recently become available on the Internet. However, these freely-distributable databases are not accessible via a uniform interface, and are not accessible from a single site. They are often small and incomplete individually, but would collectively provide an interesting and useful database of English words. Examples include the Jargon file, the WordNet database, MICRA`s version of the 1913 Webster`s Revised Unabridged Dictionary, and the Free Online Dictionary of Computing. (See the DICT protocol specification (RFC) for references.) Translating and non-English dictionaries are also becoming available (for example, the FOLDOC dictionary is being translated into Spanish). The webster protocol is not suitable for providing access to a large number of separate dictionary databases, and extensions to the current webster protocol were not felt to be a clean solution to the dictionary database problem. The DICT protocol is designed to provide access to multiple databases. Word definitions can be requested, the word index can be searched (using an easily extended set of algorithms), information about the server can be provided (e.g., which index search strategies are supported, or which databases are available), and information about a database can be provided (e.g., copyright, citation, or distribution information). Further, the DICT protocol has hooks that can be used to restrict access to some or all of the databases. **dictd**(8) is a server that implements the DICT protocol. Bret Martin implemented another server, and several people (including Bret and myself) have implemented clients in a variety of languages.

## OPTIONS

**-V or --version**
Display version information.

**--license**
Display copyright and license information.

**-h or --help**
Display help information.

**-v or --verbose or -dverbose**
Be verbose.

**-c *file* or --config *file***
Specify configuration file. The default is */etc/dictd.conf*, but may be changed in the *dictd.h* file at compile time (DICT_CONFIG_FILE).

**-p *port* or --port *port***
Specifies the port (e.g., 2628). The default is 2628, as specified in the DICT Protocol RFC, but may be changed in the *dictd.h* file at compile time (DICT_DEFAULT_SERVICE).

**--depth *length***
Specify the queue length for **listen**(2). Specifies the number of pending socket connections which are queued by the operating system. Some operating systems may silently limit this value to 5 (older BSD systems) or 128 (Linux). The default is 10 but may be changed in the *dictd.h* file at compile time (DICT_QUEUE_DEPTH).

**--delay *seconds***
Specifies the number of seconds a client may be idle before the server will close the connection. Idle time is defined to be the time the server is waiting for input and does not include the time the server spends searching the database. Connections are closed without warning since no provision for premature connection termination is specified in the DICT protocol RFC. The default is 600 seconds (10 minutes), but may be changed in the *dictd.h* file at compile time (DICT_DEFAULT_DELAY).

**--facility *facility***
Specifies the syslog facility to use. The use of this option implies the **-s** option to turn on logging via syslog. When the operating system libraries support SYSLOG_NAMES, the names used for this option should be those listed in **syslog.conf**(5). Otherwise, the following names are used (assuming the particular facility is defined in the header files): auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp, local0, local1, local2, local3, local4, local5, local6, and local7.

**-f or --force**
Force the daemon to start even if an instance of the daemon is already running. (This is of little value unless a non-default port is specified with **-p**, since, if one instance is bound to a port, the second one fails when it can not bind to the port.)

**--limit *children***
Specifies the number of daemons that may be running simultaneously. Each daemon services a single connection. If the limit is exceeded, a (serialized) connection will be made by the server process, and a response code 420 (server temporarily unavailable) will be sent to the client. This parameter should be adjusted to prevent the server machine from being overloaded by dict clients, but should not be set so low that many clients are denied useful connections. The default is 100, but may be changed in the *dictd.h* file at compile time (DICT_DAEMON_LIMIT).

**--locale *locale***
Specifies the locale used for searching. If no locale is specified, the "C" locale is used. The locale used for the server should be the same as that used for dictfmt when the database was built

(specifically, the locale under which the index was sorted). The locale should be specified for both 8-bit and UTF-8 formats. If locale contains utf8 or utf-8 substring, UTF-8 format is expected. Note that if your database is not in ASCII7 or UTF-8 format, then the dictd server will not be compliant to RFC 2229.

**-s**
: Log using the **syslog**(3) facility.

**-L *file* or --logfile *file***
: Specify the file for logging. The filename specified is recomputed on each use using the **strftime**(3) call. For example, a filename ending in ".%Y%m%d" will write to log files ending in the year, month, and date that the log entry was written. **NOTE:** If **dictd** does not have write permission for this file, it will silently fail.

**-m *minutes* or --mark minutes**
: How often a timestamp should be logged. (This is effective only if logging has been enabled with the -s or -L option, or with a debugging option.)

**--default-strategy *strategy***
: Set the default strategy for MATCH search type. The default is `lev`.

**--without-strategy *strat1,strat2,...***
: Disable specified strategies. By default all search strategies are enabled.

**--add-strategy *strat:descr***
: Adds strategy `strat` with the description `descr`. A new search strategy may be implemented with a help of plugins.

**--no-mmap**
: do not use the mmap() function and read entire files into memory instead.

**--test *word* or -t word**
: self test -- lookup word

**--test-file *file* or --ftest file**
: self test -- lookup all words in file

**--test-strategy *strategy***
: self test -- set search strategy for --test and --ftest. The default is `exact`.

**--test-db *database***
: self test -- set dictionary to be searched. The default is `*`.

**--test-match**
: self test -- set search type to MATCH. The default is DEFINE.

**-l** *option* or **--log** *option* Specify a logging option. This is effective only if logging has been enabled with the **-s** or **-L** option, or logging to the console has been activated with a debugging option (e.g., **--debug nodetach**. Only one option may be set with each invocation of this option; however, multiple invocations of this option may be made in one dictd command line. For instance:
dictd -s --log stats --log found --log notfound
is a valid command line, and sets three logging options.

Some of the more verbose logging options are used primarily for debugging the server code, and are not practical for normal use.

**server**
: Log server diagnostics. This is extremely verbose.

**connect**
: Log all connections.

**stats**
: Log all children terminations.

**command**
: Log all commands. This is extremely verbose.

**client**
: Log results of CLIENT command.

**found**
: Log all words found in the databases.

**notfound**
: Log all words not found in the databases.

**timestamp**
: When logging to a file, use a full timestamp like that which syslog would produce. Otherwise, no timestamp is made, making the files shorter.

**host**
: Log name of foreign host.

**auth**
: Log authentication failures.

**min**
: Set a minimal number of options. If logging is activated (to a file, or via syslog), and no options are set, then the minimal set of options will be used. If options are set, then only those options specified will be used.

**all**
: Set all of the options.

**none**
: Clear all of the options. To facilitate location of interesting information in the log file, entries are marked with initial letters indicating the class of the line being logged:

**I**
: Information about the server, connections, or termination statistics. These lines are generally not designed to be parsed automatically.

**E**
: Error messages.

**C**
: CLIENT command information.

**D**
: Definitions found in the databases searched.

**M**
: Matches found in the database searched.

**N**
: Matches which were not found in the databases searched.

**T**
: Trace of exact line sent by client.

**A**

Authentication information. To preserve anonymity of the client, do *not* use the **connect** or **host** options. Clients may or may not send host information using the CLIENT command, but this should be an option that is selectable on the client side.

**-d *option***

Activate a debugging option. There are several, all of which are only useful to developers. They are documented here for completeness. A list can be obtained interactively by using **-d** with an illegal option.

**verbose**

The same as **-v** or **--verbose**. Adds verbosity to other options.

**scan**

Debug the scanner for the configuration file.

**parse**

Debug the parser for the configuration file.

**search**

Debug the character folding and binary search routines.

**init**

Report database initialization.

**port**

Log client-side port number to the log file.

**lev**

Debug Levenshtein search algorithm.

**auth**

Debug the authorization routines.

**nodetach**

Do not detach as a background process. Implies that a copy of the log file will appear on the standard output.

**nofork**

Do not fork daemons to service requests. Be a single-threaded server. This option implies **nodetach**, and is most useful for using a debugger to find the point at which daemon processes are dumping core.

**alt**

Debugs **altcompare** in *index.c*.

## CONFIGURATION FILE

### Introduction

The configuration file defaults to */etc/dictd.conf*, but can be specified on the command line with the **-c** option (see above). The configuration file has four distinct sections. At this time, each section must appear in the specified order, although only the Database section is required.

The file is divided up into different sections. The Site Section should come first, followed by the Access Section, the Database Section, and the User Section. Sections are optional, but they should be in the order listed here.

**Syntax**

The following keywords are valid in a configuration file: access, allow, deny, group, database, data, index, filter, prefilter, postfilter, name, include, user, authonly, site. Keywords are case sensitive. String arguments that contain spaces should be surrounded by double quotes. Without quoting, strings may contain alphanumeric characters and _, -,
 ., and *, but not spaces. Strings can be continued between lines. ", \, , <NL> are treated as double quote, backslash, new line and no symbol respectively. Comments start with # and extend to the end of the line.

**Site Section**

**site *string***

Used to specify the filename for the site information file, a flat text file which will be displayed in response to the SHOW SERVER command. This section, if present, must be first.

**Access Section**

**access { *access specification* }**

This section, the second if the Site Section is present, contains access restrictions for the server and all of the databases collectively. Per-database control is specified in the Database Section.

**Database Section**

**database *string* { *database specification* }**

The string specifies the name of the database (e.g., wn or web1913). (This is an arbitrary name selected by the administrator, and is not necessarily related to the file name or any name listed in the data file. A short, easy to type name is often selected for easy use with **dict -d.)**

**NOTE:** If the files specified in the database specification do not exist on the system, dictd may silently fail.

**database_virtual *string* { *virtual database specification* }**

This section specifies the virtual database. The string specifies the name of the database (e.g., en-ru or fren).

**database_plugin *string* { *plugin specification* }**

This section specifies the plugin. The string specifies the name of the database.

**database_exit**

Excludes following databases from the \`*\` database. By default \`*\` means all databases available. Look at \`example_virtual.conf\` file for example configuration.

**NOTE:** If you use \`virtual\` dictionaries, you should use this directive, otherwise you will search the same dictionary twice.

**User Section**

**user *string* string**

The first string specifies the username, and the second string specifies the shared secret for this username. When the AUTH command is used, the client will provide the username and a hashed version of the shared secret. If the shared secret matches, the user is said to have authenticated, and will have access to databases whose access specifications allow that user (by name, or by wildcard). If present, this section must appear last in the configuration file. There may be many user entries. The shared secret should be kept secret, as anyone who has access to it can access the shared databases (assuming access is not denied by domain name).

**Access Specification**

Access specifications may occur in the Access Section or in the Database Section. The access specification will be described here. For allow, deny, and authonly, a star (*) may be used as a wild card that matches any number of characters. A question mark (?) may be used as a wildcard that matches a single character. For example, 10.0.0.* and *.edu are valid strings. Further, a range of IP addresses and an IP address followed by a netmask may be specified. For example, 10.0.0.0:10.0.0.255, 10.0.0.0/24, and 10.0.0.* all specify the same range of IP numbers. Notation cannot be combined on the same line. If the notation does not make sense, access will be denied by default. Use the *--debug auth* option to debug related problems. Note that these specifications take only one string per specification line. However, you can have multiple lines of each type. The syntax is as follows:

**allow *string***

The string specifies a domain name or IP address which is allowed access to the server (in the Access Section) or to a database (in the Database Section). Note that more than one string is not permitted for a single "allow" line, but more than one "allow" lines are permitted in the configuration file.

**deny *string***

The string specifies a domain name or IP address which is denied access to the server (in the Access Section) or to a database (in the Database Section). Note that if reverse DNS is not working, then only the IP number will be checked. Therefore, it is essential to deny networks based on IP number, since a denial based on domain name may not always be checked.

**authonly *string***

This form is only useful in the Access Section. The string specifies a domain name or IP address which is allowed access to the server but not to any of the databases. All commands are valid except DEFINE, MATCH, and SHOW DB. More specifically AUTH is a valid command, and commands which access the databases are not allowed.

**user *string***

This form is only useful in the Database Section. The string specifies a username that is allowed to access this database after a successful AUTH command is executed.

**Database Specification**

The database specification describes the database:

**data *string***

Specifies the filename for the flat text database. If the filename does not begin with `.` or `/`, it is prepended with $datadir/. It is a compile time option. You can change this behaviour by editing Makefile or running ./configure --datadir=...

**index *string***

Specifies the filename for the index file. Path matter is similar to that described above in "data" option .

**index_suffix *string***

This is optional index file to make `suffix` search strategy faster (binary search). It is generated by `dictfmt_index2suffix`. Run "dictfmt_index2suffix --help" for more information. Path matter is similar to that described above in "data" option .

**index_word *string***

This is optional index file to make `word` search strategy faster (binary search). It is generated by `dictfmt_index2word`. Run "dictfmt_index2word --help" for more information. Path matter is similar to that described above in "data" option .

**prefilter *string***

Specifies the prefilter command. When a chunk of the compressed database is read, it will be filtered with this filter before being decompressed. This may be used to provide some additional compression that knows about the data and can provide better compression than the LZ77 algorithm used by zlib.

**postfilter *string***

Specifies the postfilter command. When a chunk of the compressed database is read, it will be filtered with this filter before the offset and length for the entry are used to access data. This is provided for symmetry with the prefilter command, and may also be useful for providing additional database compression.

**filter *string***

Specifies the filter command. After the entry is extracted from the database, it will be filtered with this filter. This may be used to provide formatting for the entry (e.g., for html). **Warning:** This is not currently implemented.

**name *string***

Specifies the short name of the database (e.g., "1913 Webster`s"). If the string begins with @, then it specifies the headword to look up in the dictionary to find the short name of the database. The default is "@00-database-short", but this may be changed in the *dictd.h* file at compile time (DICT_SHORT_ENTRY_NAME).

**info *string***

Specifies the information about database. If the string begins with @, then it specifies the headword to look up in the dictionary to find information. The default is "@00-database-info", but this may be changed in the *dictd.h* file at compile time (DICT_INFO_ENTRY_NAME).

**invisible**

Makes dictionary invisible to the clients i.e. this dictionary will not be recognized or shown by DEFINE, MATCH, SHOW INFO, SHOW SERVER and SHOW DB commands. If some definitions or matches are found in invisible dictionary, the name of the upper visible virtual dictionary or `*` is returned. **NOTE:** There is no sense to make dictionary invisible unless it is included to the virtual dictionary.

**Virtual Database Specification**

The virtual database specification describes the virtual database:

**database_list *string***

Specifies a list of databases which are included into the virtual database. Database names are in the string and are separated by comma.

**name *string***

Specifies the short name of the database. String beginning with `@` symbol is not treated as an entry name.

**info *string***

Specifies the information about database. String beginning with `@` symbol is not treated as an entry name.

**invisible**

Makes dictionary invisible to the clients. See *database specification*

**NOTE:**

Another way to implement a virtual database is to create database files by dictfmt_virtual executable

**Plugin Specification**

**plugin *string***

Specifies a filename of the plugin.

**data *string***

Specifies data for initializing plugin.

**name *string***

Specifies the short name of the database. See *database specification*

**info *string***

Specifies the information about database. See *database specification*

**invisible**

Makes dictionary invisible to the clients. See *database specification*

**NOTE:**

Another way to configure plugin is to create database files by dictfmt_plugin executable

**include *string***

The text of the file "string" (usually a database specification) will be read as if it appeared at this location in the configuration file. Nested includes are not permitted.

## DETERMINATION OF ACCESS LEVEL

When a client connects, the global access specification is scanned, in order, until a specification matches. If no access specification exists, all access is allowed (e.g., the action is the same as if "allow *" was the only item in the specification). For each item, both the hostname and IP are checked. For example, consider the following access specification:

```
allow 10.42.*
authonly *.edu
deny *
```

With this specification, all clients in the 10.42 network will be allowed access to unrestricted databases; all clients from *.edu sites will be allowed to authenticate, but will be denied access to all databases, even those which are otherwise unrestricted; and all other clients will have their connection terminated immediately. The 10.42 network clients can send an AUTH command and gain access to restricted databases. The *.edu clients must send an AUTH command to gain access to any databases, restricted or unrestricted. When the AUTH command is sent, the access list for each database is scanned, in order, just as the global access list is scanned. However, after authentication, the client has an associated username. For example, consider the following access specification:

```
user u1
deny *.com
user u2
allow *
```

If the client authenticated as u1, then the client will have access to this database, even if the client comes from a *.com site. In contrast, if the client authenticated as u2, the client will only have access if it does not come from a *.com site. In this case, the "user u2" is redundant, since that client would also match "allow *". **Warning:** Checks are performed for domain names and for IP addresses. However, if reverse DNS for a specific site is not working, it is possible that a domain name may not be available for checking. Make sure that all denials use IP addresses. (And consider a future enhancement: if a domain name is not available, should denials that depend on a domain name match anything? This is the more conservative viewpoint, but it is not currently implemented.)

## SEARCH ALGORITHMS

The DICT standard specifies a few search algorithms that must be implemented, and permits others to be supported on a server-dependent basis. The following search strategies are supported by this server. Note that *all* strategies are case insensitive. Most ignore non-alphanumeric, non-whitespace characters.

**exact**

An exact match. This algorithm uses a binary search and is one of the fastest search algorithms available.

**lev**
The Levenshtein algorithm (string edit distance of one). This algorithm searches for all words which are within an edit distance of one from the target word. An "edit" means an insertion, deletion, or transposition. This is a rapid algorithm for correcting spelling errors, since many spelling errors are within a Levenshtein distance of one from the original word.

**prefix**
Prefix match. This algorithm also uses a binary search and is very fast.

**re**
POSIX 1003.2 (modern) regular expression search. Modern regular expressions are the ones used by **egrep**(1). These regular expressions allow predefined character classes (e.g., [[:alnum:]], [[:alpha:]], [[:digit:]], and [[:xdigit:]] are useful for this application); uses * to match a sequence 0 or more matches of the previous atom; uses + to match a sequence of 1 or more matches of the previous atom; uses ? to match a sequence of 0 or 1 matches of the previous atom; used ^ to match the beginning of a word, uses $ to match the end of a word, and allows nested subexpression and alternation with () and |. For example, "(foo|bar)" matches all words that contain either "foo" or "bar". To match these special characters, they must be quoted with two backslashes (due to the quoting characteristics of the server). **Warning:** Regular expression matches can take 10 to 300 times longer than substring matches. On a busy server, with many databases, this can required more than 5 minutes of waiting time, depending on the complexity of the regular expression.

**regexp**
Old (basic) regular expressions. These regular expressions don`t support |, +, or ?. Groups use escaped parentheses. While modern regular expressions are generally easier to use, basic regular expressions have a back reference feature. This can be used to match a second occurrence of something that was already matched. For example, the following expression finds all words that begin and end with the same three letters:

```
^\(...\).*\1$
```

Note the use of the double backslashes to escape the special characters. This is required by the DICT protocol string specification (a single backslash quotes the next character -- we use two to get a single backslash through to the regular expression engine). **Warning:** Note that the use of backtracking is even slower than the use of general regular expressions.

**soundex**
The Soundex algorithm, a classic algorithm for finding words that sound similar to each other. The algorithm encodes each word using the first letter of the word and up to three digits. Since the first letter is known, this search is relatively fast, and it sometimes good for correcting spelling errors when the Levenshtein algorithm doesn`t help.

**substring**
Match a substring anywhere in the headword. This search strategy uses a modified Boyer-Moore-Horspool algorithm. Since it must search the whole index file, it is not as fast as the exact and prefix matches.

**suffix**
Suffix match. This search strategy also uses a modified Boyer-Moore-Horspool algorithm, and is as fast as the substring search. If the optional index_suffix string file is listed in the configuration file this search is much faster.

**word**
Match any single word, even if part of a multi-word entry. If the optional index_word string file is listed in the configuration file this search is much faster.

## DATABASE FORMAT

Databases for **dictd** are distributed separately. A database consists of two files. One is a flat text file, the other in the index. The flat text file contains dictionary entries (or any other suitable data), and the index contains tab-delimited tuples consisting of the headword, the byte offset at which this entry begins in the flat text file, and the length of the entry in bytes. The offset and length are encoded using base 64 encoding using the 64-character subset of International Alphabet IA5 discussed in RFC 1421 (printable encoding) and RFC 1522 (base64 MIME). Encoding the offsets in base 64 saves considerable space when compared with the usual base 10 encoding, while still permitting tab characters (ASCII 9) to be used for delimiting fields in a record. Each record ends with a newline (ASCII 10), so the index file is human readable. The flat text file may be compressed using **gzip**(1) (not recommended) or **dictzip**(1) (highly recommended). Optimal speed will be obtained using an uncompressed file. However, the **gzip** compression algorithm works very well on plain text, and can result in space savings typically between 60 and 80%. Using a file compressed with **gzip**(1) is not recommended, however, because random access on the file can only be accomplished by serially decompressing the whole file, a process which is prohibitively slow. **dictzip**(1) uses the same compression algorithm and file format as does **gzip**(1), but provides a table that can be used to randomly access compressed blocks in the file. The use of 50-64kB blocks for compression typically degrades compression by less than 10%, while maintaining acceptable random access capabilities for all data in the file. As an added benefit, files compressed with **dictzip**(1) can be decompressed with **gzip**(1) or **zcat**(1). (Note: recompressing a **dictzip**`d file using, for example, **znew**(1) will destroy the random access characteristics of the file. Always compress data files using **dictzip**(1).)

## ACKNOWLEDGEMENTS

Special thanks to Jean-loup Gailly and Mark Adler for writing the **zlib** general purpose data compression library. The version contained with **dictd** is not necessarily an original version and **may have been modified** (unnecessary files may have been deleted to make the distribution smaller; makefiles may have been modified to ease compilation; see zlib/README.DICT for any significant changes). For more information on **zlib**, please see the **zlib** home page at

*http://www.gzip.org/zlib/*

The key features of the **dictzip** random-access compression algorithm utilize a documented extension of the gzip format, and do not require any modifications to **zlib**. Special thanks to Henry Spencer for his regex package. The package contained with **dictd** is not necessarily an original version and **may have been modified** (unnecessary files may have been deleted to make the distribution smaller; makefiles may have been modified to ease compilation; see regex/README.DICT for any significant changes). For more information on regex, please see

*ftp://zoo.toronto.edu/pub/regex.shar*

## COPYING

The main source files for the **dictd** server and the **dictzip** compression program were written by Rik Faith (faith@dict) and are distributed under the terms of the GNU General Public License. If you need to distribute under other terms, write to the author. The main libraries used by these programs (zlib, regex, libmaa) are distributed under different terms, so you may be able to use the libraries for applications which are incompatible with the GPL -- please see the copyright notices and license information that come with the libraries for more information, and consult with your attorney to resolve these issues.

## BUGS

The regular expression searches do not ignore non-whitespace, non-alphanumeric characters as do the other searches. In practice, this isn`t much of a problem. The databases are memory mapped and cannot be updated while the server is running. There is no way to get a running server to re-read the configuration file, so databases cannot be added or deleted on the fly.

## FILES

*/etc/dictd.conf*
*/usr/sbin/dictd*

## SEE ALSO

**dictfmt**(1), **dictfmt_virtual**(1), **dict**(1), **dictzip**(1), **gunzip**(1), **zcat**(1), **webster**(1), **RFC 2229**