BIND 8 is much more configurable than previous release of BIND. There are entirely new areas of configuration, such as access control lists and categorized logging. Many options that previously applied to all zones can now be used selectively. These features, plus a consideration of future configuration needs led to the creation of a new configuration file format.

A BIND 8 configuration consists of two general features, statements and comments. All statements end with a semicolon. Many statements can contain substatements, which are each also terminated with a semicolon.

The following statements are supported: specifies what the server logs, and where the log messages are sent

controls global server configuration options and sets defaults for other statements

defines a zone

defines a named IP address matching list, for access control and other uses

specifies key information for use in authentication and authorization

defines DNSSEC keys that are preconfigured into the server and implicitly trusted

sets certain configuration options for individual remote servers

declares control channels to be used by the utility

includes another file

The and statements may only occur once per configuration, while the rest may appear numerous times. Further detail on each statement is provided in individual sections below.

Comments may appear anywhere that whitespace may appear in a BIND configuration file. To appeal to programmers of all kinds, they can be written in C, C++, or shell/perl constructs.

C-style comments start with the two characters (slash, star) and end with (star, slash). Because they are completely delimited with these characters, they can be used to comment only a portion of a line or to span multiple lines.

C-style comments cannot be nested. For example, the following is not valid because the entire comment ends with the first

```
/* This is the start of a comment.
   This is still part of the comment.  /* This is an incorrect attempt at nesting a comment. */
   This is no longer in any comment. */
```

C++-style comments start with the two characters (slash, slash) and continue to the end of the physical line. They cannot be continued across multiple physical lines; to have one logical comment span multiple lines, each line must use the pair. For example:

// This is the start of a comment. The next line // is a new comment, even though it is logically // part of the previous comment.

Shell-style (or perl-style, if you prefer) comments start with the character (hash or pound or number or octothorpe or whatever) and continue to the end of the physical line, like C++ comments. For example:

This is the start of a comment. The next line # is a new comment, even though it is logically # part of the previous comment.

you cannot use the (semicolon) character to start a comment such as you would in a zone file. The semicolon indicates the end of a configuration statement, so whatever follows it will be interpreted as the start of the next statement.

BIND 4.9.x configuration files can be converted to the new format by using a shell script that is part of the BIND 8.2.x source kit.

Described below are elements used throughout the BIND configuration file documentation. Elements which are only associated with one statement are described only in the section describing that statement.

The name of an as defined by the statement.

A list of one or more or elements, as described in the section.

One or more integers valued 0 through 255 separated only by dots (“.”), such as or

A quoted string which will be used as a DNS name, for example

A quoted string which will be used as a pathname, such as

An IP address in with exactly four elements in notation.

An IP port through with values below 1024 typically restricted to root-owned processes. In some cases an asterisk (“*”) character can be used as a placeholder to select a random high-numbered port.

An IP network specified in form, followed by “/” and then the number of bits in the netmask. E.g. is the network with netmask is network with netmask

A string representing the name of a shared key, to be used for transaction security.

A non-negative integer with an entire range limited by the range of a C language signed integer (2,147,483,647 on a machine with 32 bit integers). Its acceptable value might further be limited by the context in which it is used.

A the word or the word

The maximum value of is that of unsigned long integers on the machine. requests unlimited use, or the maximum available amount. uses the limit that was in force when the server was started.

A can optionally be followed by a scaling factor: or for kilobytes, or for megabytes, and or for gigabytes, which scale by 1024, 1024*1024, and 1024*1024*1024 respectively.

Integer storage overflow is currently silently ignored during conversion of scaled values, resulting in values less than intended, possibly even negative. Using is the best way to safely set a really large number.

Either or The words and are also accepted, as are the numbers

address_match_list = 1**address_match_element*

address_match_element = ["!"] (*address_match_list* /
ip_address / *ip_prefix* /
acl_name / "key " *key_id*) ";"

Address match lists are primarily used to determine access control for various server operations. They are also used to define priorities for querying other nameservers and to set the addresses on which will listen for queries. The elements which constitute an address match list can be any of the following:

an (in notation, an (in the ’/’-notation), A as defined by the statement, the name of an address match list previously defined with the statement, or another

Elements can be negated with a leading exclamation mark (“!”), and the match list names and are predefined. More information on those names can be found in the description of the statement.

The addition of the clause made the name of this syntactic element something of a misnomer, since security keys can be used to validate access without regard to a host or network address. Nonetheless, the term “address match list” is still used throughout the documentation.

When a given IP address or prefix is compared to an address match list, the list is traversed in order until an element matches. The interpretation of a match depends on whether the list is being used for access control, defining ports, or as a topology, and whether the element was negated.

When used as an access control list, a non-negated match allows access and a negated match denies access. If there is no match at all in the list, access is denied. The clauses and all use address match lists like this. Similarly, the option will cause the server to not accept queries on any of the machine's addresses which do not match the list.

When used with the option, a non-negated match returns a distance based on its position on the list (the closer the match is to the start of the list, the shorter the distance is between it and the server). A negated match will be assigned the maximum distance from the server. If there is no match, the address will get a distance which is further than any non-negated list element, and closer than any negated element.

Because of the first-match aspect of the algorithm, an element that defines a subset of another element in the list should come before the broader element, regardless of whether either is negated. For example, in the 1.2.3.13 element is completely useless, because the algorithm will match any lookup for 1.2.3.13 to the 1.2.3/24 element. Using fixes that problem by having 1.2.3.13 blocked by the negation but all other 1.2.3.* hosts fall through.

```
logging {
 [ channel channel_name {
  ( file path_name
     [ versions ( number | unlimited ) ]
     [ size size_spec ]
   | syslog ( kern | user | mail | daemon | auth | syslog | lpr |
            news | uucp | cron | authpriv | ftp |
            local0 | local1 | local2 | local3 |
            local4 | local5 | local6 | local7 )
   | null );

  [ severity ( critical | error | warning | notice |
            info  | debug [ level ] | dynamic ); ]
  [ print-category yes_or_no; ]
  [ print-severity yes_or_no; ]
  [ print-time yes_or_no; ]
 }; ]

 [ category category_name {
  channel_name; [ channel_name; ... ]
 }; ]
 ...  };
```

The statement configures a wide variety of logging options for the nameserver. Its phrase associates output methods, format options and severity levels with a name that can then be used with the phrase to select how various classes of messages are logged.

Only one statement is used to define as many channels and categories as are wanted. If there are multiple logging statements in a configuration, the first defined determines the logging, and warnings are issued for the others. If there is no logging statement, the logging configuration will be:

```
logging {
   category default { default_syslog; default_debug; };
   category panic { default_syslog; default_stderr; };
   category packet { default_debug; };
   category eventlib { default_debug; };
};
```

The logging configuration is established as soon as the statement is parsed. If you want to redirect messages about processing of the entire configuration file, the statement must appear first. Even if you do not redirect configuration file parsing messages, we recommend always putting the statement first so that this rule need not be consciously recalled if you ever do need want the parser's messages relocated.

All log output goes to one or more ‘‘channels’’; you can make as many of them as you want.

Every channel definition must include a clause that says whether messages selected for the channel go to a file, to a particular syslog facility, or are discarded. It can optionally also limit the message severity level that will be accepted by the channel (default is and whether to include a time stamp generated by the category name, or severity level. The default is not to include any of those three.

The word as the destination option for the channel will cause all messages sent to it to be discarded; other options for the channel are meaningless.

The clause can include limitations both on how large the file is allowed to become, and how many versions of the file will be saved each time the file is opened.

The option for files is simply a hard ceiling on log growth. If the file ever exceeds the size, then will just not write anything more to it until the file is reopened; exceeding the size does not automatically trigger a reopen. The default behavior is to not limit the size of the file.

If you use the logfile option, then will retain that many backup versions of the file by renaming them when opening. For example, if you choose to keep 3 old versions of the file lamers.log then just before it is opened lamers.log.1 is renamed to lames.log.2, lamers.log.0 is renamed to lamers.log.1, and lamers.log is renamed to lamers.log.0. No rolled versions are kept by default; any existing log file is simply appended. The keyword is synonymous with in current BIND releases. Example usage of size and versions options:

```
channel an_example_level {
   file "lamers.log" versions 3 size 20m;
   print-time yes;
   print-category yes;
};
```

The argument for the clause is a syslog facility as described in the manual page. How will handle messages sent to this facility is described in the manual page. If you have a system which uses a very old version of syslog that only uses two arguments to the function, then this clause is silently ignored.

The clause works like syslog's ‘‘priorities’’, except that they can also be used if you are writing straight to a file rather than using syslog. Messages which are not at least of the severity level given will not be selected for the channel; messages of higher severity levels will be accepted.

If you are using syslog, then the priorities will also determine what eventually passes through. For example, defining a channel facility and severity as and but only logging via will cause messages of severity and to be dropped. If the situation were reversed, with writing messages of only or higher, then would print all messages it received from the channel.

The server can supply extensive debugging information when it is in debugging mode. If the server's global debug level is greater than zero, then debugging mode will be active. The global debug level is set either by starting the server with the flag followed by a positive integer, or by sending the running server the signal (for example, by using The global debug level can be set to zero, and debugging mode turned off, by sending the server the signal (as with All debugging messages in the server have a debug level, and higher debug levels give more more detailed output. Channels that specify a specific debug severity, e.g.

```
channel specific_debug_level {
   file "foo";
   severity debug 3;
};
```

will get debugging output of level 3 or less any time the server is in debugging mode, regardless of the global debugging level. Channels with severity use the server's global level to determine what messages to print.

If has been turned on, then the date and time will be logged. may be specified for a syslog channel, but is usually pointless since syslog also prints the date and time. If is requested, then the category of the message will be logged as well. Finally, if is on, then the severity level of the message will be logged. The options may be used in any combination, and will always be printed in the following order: time, category, severity. Here is an example where all three options are on:

```
28-Apr-1997 15:05:32.863 default: notice: Ready to answer queries.
```

There are four predefined channels that are used for default logging as follows. How they are used used is described in the next section,

```
channel default_syslog {
    syslog daemon;      # send to syslog's daemon facility
    severity info;      # only send priority info and higher
};

channel default_debug {
    file "named.run";   # write to named.run in the working directory
                        # Note: stderr is used instead of "named.run"
                        # if the server is started with the -f option.
    severity dynamic;   # log at the server's current debug level
};

channel default_stderr { # writes to stderr
    file "<stderr>";    # this is illustrative only; there's currently
                        # no way of specifying an internal file
                        # descriptor in the configuration language.
    severity info;      # only send priority info and higher
};

channel null {
    null;               # toss anything sent to this channel
};
```

Once a channel is defined, it cannot be redefined. Thus you cannot alter the built-in channels directly, but you can modify the default logging by pointing categories at channels you have defined.

There are many categories, so you can send the logs you want to see wherever you want, without seeing logs you don't want. If you don't specify a list of channels for a category, then log messages in that category will be sent to the category instead. If you don't specify a default category, the following ''default default'' is used:

```
category default { default_syslog; default_debug; };
```

As an example, let's say you want to log security events to a file, but you also want keep the default logging behavior. You'd specify the following:

```
channel my_security_channel {
    file "my_security_file";
    severity info;
};
category security { my_security_channel;
                  default_syslog; default_debug; };
```

To discard all messages in a category, specify the channel:

```
category lame-servers { null; };
category cname { null; };
```

The following categories are available:

The catch-all. Many things still aren't classified into categories, and they all end up here. Also, if you don't specify any channels for a category, the default category is used instead. If you do not define the default category, the following definition is used:

High-level configuration file processing.

Low-level configuration file processing.

A short log message is generated for every query the server receives.

Messages like ''Lame server on ...''

Statistics.

If the server has to shut itself down due to an internal problem, it will log the problem in this category as well as in the problem's native category. If you do not define the panic category, the following definition is used:

Dynamic updates.

Negative caching.

Zone transfers the server is receiving.

Zone transfers the server is sending.

All database operations.

Debugging info from the event system. Only one channel may be specified for this category, and it must be a file channel. If you do not define the eventlib category, the following definition is used:

Dumps of packets received and sent. Only one channel may be specified for this category, and it must be a file channel. If you do not define the packet category, the following definition is used:

The NOTIFY protocol.

Messages like ‘‘... points to a CNAME’’.

Approved/unapproved requests.

Operating system problems.

Internal consistency check failures.

Periodic maintenance events.

Zone loading messages.

Messages arising from response checking, such as ‘‘Malformed response ...’’, ‘‘wrong ans. name ...’’, ‘‘unrelated additional info ...’’, ‘‘invalid RR type ...’’, and ‘‘bad referral ...’’.

options {
 [version *version_string*;]
 [directory *path_name*;]
 [named-xfer *path_name*;]
 [dump-file *path_name*;]
 [memstatistics-file *path_name*;]
 [pid-file *path_name*;]
 [statistics-file *path_name*;]
 [auth-nxdomain *yes_or_no*;]
 [deallocate-on-exit *yes_or_no*;]
 [dialup *yes_or_no*;]
 [fake-iquery *yes_or_no*;]
 [fetch-glue *yes_or_no*;]
 [has-old-clients *yes_or_no*;]
 [host-statistics *yes_or_no*;]
 [multiple-cnames *yes_or_no*;]
 [notify *yes_or_no*;]
 [recursion *yes_or_no*;]
 [rfc2308-type1 *yes_or_no*;]
 [use-id-pool *yes_or_no*;]
 [treat-cr-as-space *yes_or_no*;]
 [also-notify *yes_or_no*;]
 [forward (only | first);]
 [forwarders { [*in_addr* ; [*in_addr* ; ...]] };]
 [check-names (master | slave | response) (warn | fail | ignore);]
 [allow-query { *address_match_list* };]
 [allow-recursion { *address_match_list* };]
 [allow-transfer { *address_match_list* };]
 [blackhole { *address_match_list* };]
 [listen-on [port *ip_port*] { *address_match_list* };]

[query-source [address (*ip_addr* | *)]
[port (*ip_port* | *)] ;]
[lame-ttl *number*;]
[max-transfer-time-in *number*;]
[max-ncache-ttl *number*;]
[min-roots *number*;]
[serial-queries *number*;]
[transfer-format (one-answer | many-answers);]
[transfers-in *number*;]
[transfers-out *number*;]
[transfers-per-ns *number*;]
[transfer-source *ip_addr*;]
[maintain-ixfr-base *yes_or_no*;]
[max-ixfr-log-size *number*;]
[coresize *size_spec* ;]
[datasize *size_spec* ;]
[files *size_spec* ;]
[stacksize *size_spec* ;]
[cleaning-interval *number*;]
[heartbeat-interval *number*;]
[interface-interval *number*;]
[statistics-interval *number*;]
[topology { *address_match_list* };]
[sortlist { *address_match_list*|*fR };]*
[rrset-order { order_spec ; [*order_spec* ; ... [[}; };

The options statement sets up global options to be used by BIND. This statement may appear at only once in a configuration file; if more than one occurrence is found, the first occurrence determines the actual options used, and a warning will be generated. If there is no options statement, an options block with each option set to its default will be used.

The version the server should report via the ndc command or via a query of name in class chaos. The default is the real version number of ths server, but some server operators prefer the string ( ).

The working directory of the server. Any non-absolute pathnames in the configuration file will be taken as relative to this directory. The default location for most server output files (e.g. is this directory. If a directory is not specified, the working directory defaults to the directory from which the server was started. The directory specified should be an absolute path.

The pathname to the named-xfer program that the server uses for inbound zone transfers. If not specified, the default is system dependent (e.g.).

The pathname of the file the server dumps the database to when it receives signal (as sent by). If not specified, the default is

The pathname of the file the server writes memory usage statistics to on exit, if is If not specified, the default is

The pathname of the file the server writes its process ID in. If not specified, the default is operating system dependent, but is usually or The pid-file is used by programs like that want to send signals to the running nameserver.

The pathname of the file the server appends statistics to when it receives signal (from If not specified, the default is

If then the bit is always set on responses, even if the server is not actually authoritative. The default is Do not turn off unless you are sure you know what you are doing, as some older software won't like it.

If then when the server exits it will painstakingly deallocate every object it allocated, and then write a memory usage report to the The default is because it is faster to let the operating system clean up. is handy for detecting memory leaks.

If then the server treats all zones as if they are doing zone transfers across a dial on demand dialup link, which can be brought up by traffic originating from this server. This has different effects according to zone type and concentrates the zone maintenance so that it all happens in a short interval, once every and hopefully during the one call. It also suppresses some of the normal zone maintenance traffic. The default is The option may also be specified in the statement, in which case it overrides the statement.

If the zone is a then the server will send out request to all the slaves. This will trigger the zone up to date checking in the slave (providing it supports allowing the slave to verify the zone while the call us up.

If the zone is a or then the server will suppress the zone regular zone up to date queries and only perform the when the expires.

If the server will simulate the obsolete DNS query type The default is

If (the default), the server will fetch ‘‘glue’’ resource records it doesn’t have when constructing the additional data section of a response. can be used in conjunction with to prevent the server’s cache from growing or becoming corrupted (at the cost of requiring more work from the client).

Setting the option to is equivalent to setting the following three options: and with and is order dependant.

If then statistics are kept for every host that the the nameserver interacts with. The default is turning on can consume huge amounts of memory.

If statistics are kept for every host that the nameserver interacts with. The default is turning on can consume huge amounts of memory.

If then multiple CNAME resource records will be allowed for a domain name. The default is Allowing multiple CNAME records is against standards and is not recommended. Multiple CNAME support is available because previous versions of BIND allowed multiple CNAME records, and these records have been used for load balancing by a number of sites.

If (the default), DNS NOTIFY messages are sent when a zone the server is authoritative for changes. The use of NOTIFY speeds convergence between the master and its slaves. Slave servers that receive a NOTIFY message and understand it will contact the master server for the zone and see if they need to do a zone transfer, and if they do, they will initiate it immediately. The option may also be specified in the statement, in which case it overrides the statement.

If and a DNS query requests recursion, then the server will attempt to do all the work required to answer the query. If recursion is not on, the server will return a referral to the client if it doesn’t know the answer. The default is See also above.

If the server will send NS records along with the SOA record for negative answers. You need to set this to no if you have an old BIND server using you as a forwarder that does not understand negative answers which contain both SOA and NS records or you have an old version of sendmail. The correct fix is to upgrade the broken server or sendmail. The default is

If the server will keep track of its own outstanding query ID’s to avoid duplication and increase randomness. This will result in 128KB more memory being consumed by the server. The default is

If the server will treat CR characters the same way it treats a space or tab. This may be necessary when loading zone files on a UNIX system that were generated on an NT or DOS machine. The default is

Defines a global list of IP addresses that also get sent NOTIFY messages whenever a fresh copy of the zone is loaded. This helps to ensure that copies of the zones will quickly converge on ‘‘stealth’’ servers. If an list is given in a statement, it will override the statement. When a statement is set to the IP addresses in the global list will not get sent NOTIFY messages for that zone. The default is the empty list (no global notification list).

The forwarding facility can be used to create a large site-wide cache on a few servers, reducing traffic over links to external nameservers. It can also be used to allow queries by servers that do not have direct access to the Internet, but wish to look up exterior names anyway. Forwarding occurs only on those queries for which the server is not authoritative and does not have the answer in its cache.

This option is only meaningful if the list is not empty. A value of the default, causes the server to query the forwarders first, and if that doesn’t answer the question the server will then look for the answer itself. If is specified, the server will only query the forwarders.

Specifies the IP addresses to be used for forwarding. The default is the empty list (no forwarding).

Forwarding can also be configured on a per-zone basis, allowing for the global forwarding options to be overridden in a variety of ways. You can set particular zones to use different forwarders, or have different behavior, or to not forward at all. See section for more information.

Future versions of BIND 8 will provide a more powerful forwarding system. The syntax described above will continue to be supported.

The server can check domain names based upon their expected client contexts. For example, a domain name used as a hostname can be checked for compliance with the RFCs defining valid hostnames.

Three checking methods are available:

No checking is done.

Names are checked against their expected client contexts. Invalid names are logged, but processing continues normally.

Names are checked against their expected client contexts. Invalid names are logged, and the offending data is rejected.

The server can check names three areas: master zone files, slave zone files, and in responses to queries the server has initiated. If has been specified, and answering the client's question would require sending an invalid name to the client, the server will send a response code to the client.

The defaults are:

```
check-names master fail;
check-names slave warn;
check-names response ignore;
```

may also be specified in the statement, in which case it overrides the statement. When used in a statement, the area is not specified (because it can be deduced from the zone type).

Access to the server can be restricted based on the IP address of the requesting system or via shared secret keys. See for details on how to specify access criteria.

Specifies which hosts are allowed to ask ordinary questions. may also be specified in the statement, in which case it overrides the statement. If not specified, the default is

Specifies which hosts are allowed to ask recursive questions. may also be specified in the statement, in which case it overrides the statement. If not specified, the default is to allow recursive queries from all hosts.

Specifies which hosts are allowed to receive zone transfers from the server. may also be specified in the statement, in which case it overrides the statement. If not specified, the default is to allow transfers from all hosts.

Specifies a list of addresses that the server will not accept queries from or use to resolve a query. Queries from these addresses will not be responded to.

The interfaces and ports that the server will answer queries from may be specified using the option. takes an optional port, and an address match list. The server will listen on all interfaces allowed by the address match list. If a port is not specified, port 53 will be used.

Multiple statements are allowed. For example,

```
listen-on { 5.6.7.8; };
listen-on port 1234 { !1.2.3.4; 1.2/16; };
```

will enable the nameserver on port 53 for the IP address 5.6.7.8, and on port 1234 of an address on the machine in net 1.2 that is not 1.2.3.4.

If no is specified, the server will listen on port 53 on all interfaces.

If the server doesn't know the answer to a question, it will query other nameservers. specifies the address and port used for such queries. If is or is omitted, a wildcard IP address (will be used. If is or is omitted, a random unprivileged port will be used. The default is

Note: currently applies only to UDP queries; TCP queries always use a wildcard IP address and a random unprivileged port.

Inbound zone transfers (processes) running longer than this many minutes will be terminated. The default is 120 minutes (2 hours).

The server supports two zone transfer methods. uses one DNS message per resource record transferred. packs as many resource records as possible into a message. is more efficient, but is only known to be understood by BIND 8.1 and patched versions of BIND 4.9.5. The default is may be overridden on a per-server basis by using the statement.

The maximum number of inbound zone transfers that can be running concurrently. The default value is 10. Increasing may speed up the convergence of slave zones, but it also may increase the load on the local system.

This option will be used in the future to limit the number of concurrent outbound zone transfers. It is checked for syntax, but is otherwise ignored.

The maximum number of inbound zone transfers (processes) that can be concurrently transferring from a given remote nameserver. The default value is 2. Increasing may speed up the convergence of slave zones, but it also may increase the load on the remote nameserver. may be overridden on a per-server basis by using the phrase of the statement.

determines which local address will be bound to the TCP connection used to fetch all zones transferred inbound by the server. If not set, it defaults to a system controlled value which will usually be the address of the interface ''closest to'' the remote end. This address must appear in the remote end's option for the zones being transferred, if one is specified. This statement sets the for all zones, but can be overriden on a per-zone basis by includinga statement within the zone block in the configuration file.

The server's usage of many system resources can be limited. Some operating systems don't support some of the limits. On such systems, a warning will be issued if the unsupported limit is used. Some operating systems don't support limiting resources, and on these systems a message will be logged.

Scaled values are allowed when specifying resource limits. For example, can be used instead of to specify a limit of one gigabyte. requests unlimited use, or the maximum available amount. uses the limit that was in force when the server was started. See the definition of in the section for more details.

The maximum size of a core dump. The default value is

The maximum amount of data memory the server may use. The default value is

The maximum number of files the server may have open concurrently. The default value is Note that on some operating systems the server cannot set an unlimited value and cannot determine the maximum number of open files the kernel can support. On such systems, choosing will cause the server to use the larger of the from and the value returned by If the actual kernel limit is larger than this value, use to specify the limit explicitly.

The will be used in a future release of the server to limit the size of the transaction log kept for Incremental Zone Transfer.

The maximum amount of stack memory the server may use. The default value is

The server will remove expired resource records from the cache every

minutes. The default is 60 minutes. If set to 0, no periodic cleaning will occur.

The server will perform zone maintenance tasks for all zones marked whenever this interval expires. The default is

60 minutes. Reasonable values are up to 1 day (1440 minutes). If set to 0, no zone maintenance for these zones will occur.

The server will scan the network interface list every minutes. The default is 60 minutes. If set to 0, interface scanning will only occur when the configuration file is loaded. After the scan, listeners will be started on any new interfaces (provided they are allowed by the configuration). Listeners on interfaces that have gone away will be cleaned up.

Nameserver statistics will be logged every minutes. The default is 60. If set to 0, no statistics will be logged.

All other things being equal, when the server chooses a nameserver to query from a list of nameservers, it prefers the one that is topologically closest to itself. The statement takes an address match list and interprets it in a special way. Each top-level list element is assigned a distance. Non-negated elements get a distance based on their position in the list, where the closer the match is to the start of the list, the shorter the distance is between it and the server. A negated match will be assigned the maximum distance from the server. If there is no match, the address will get a distance which is further than any non-negated list element, and closer than any negated element. For example,

```
topology {
    10/8;
    !1.2.3/24;
    { 1.2/16; 3/8; };
};
```

will prefer servers on network 10 the most, followed by hosts on network 1.2.0.0 (netmask 255.255.0.0) and network 3, with the exception of hosts on network 1.2.3 (netmask 255.255.255.0), which is preferred least of all.

The default topology is

When returning multiple RRs, the nameserver will normally return them in i.e. after each request, the first RR is put to the end of the list. As the order of RRs is not defined, this should not cause any problems.

The client resolver code should re-arrange the RRs as appropriate, i.e. using any addresses on the local net in preference to other addresses. However, not all resolvers can do this, or are not correctly configured.

When a client is using a local server, the sorting can be performed in the server, based on the client's address. This only requires configuring the nameservers, not all the clients.

The statement takes an address match list and interprets it even more specially than the statement does.

Each top level statement in the sortlist must itself be an explicit address match list with one or two elements. The first element (which may be an IP address, an IP prefix, an ACL name or nested address match list) of each top level list is checked against the source address of the query until a match is found.

Once the source address of the query has been matched, if the top level statement contains only one element, the actual primitive element that matched the source address is used to select the address in the response to move to the beginning of the response. If the statement is a list of two elements, the second element is treated like the address match list in a topology statement. Each top level element is assigned a distance and the address in the response with the minimum distance is moved to the beginning of the response.

In the following example, any queries received from any of the addresses of the host itself will get responses preferring addresses on any of the locally connected networks. Next most preferred are addresses on the 192.168.1/24 network, and after that either the 192.168.2/24 or 192.168.3/24 network with no preference shown between these two networks. Queries received from a host on the 192.168.1/24 network will prefer other addresses on that network to the 192.168.2/24 and 192.168.3/24 networks. Queries received from a host on the 192.168.4/24 or the 192.168.5/24 network will only prefer other addresses on their directly connected networks.

```
sortlist {
        { localhost;        // IF   the local host
          { localnets;      // THEN first fit on the
            192.168.1/24;   //      following nets
            { 192,168.2/24; 192.168.3/24; }; }; };
        { 192.168.1/24;     // IF   on class C 192.168.1
          { 192.168.1/24;   // THEN use .1, or .2 or .3
            { 192.168.2/24; 192.168.3/24; }; }; };
```

```
{ 192.168.2/24;     // IF   on class C 192.168.2
  { 192.168.2/24;   // THEN use .2, or .1 or .3
    { 192.168.1/24; 192.168.3/24; }; }; };
{ 192.168.3/24;     // IF   on class C 192.168.3
  { 192.168.3/24;   // THEN use .3, or .1 or .2
    { 192.168.1/24; 192.168.2/24; }; }; };
{ { 192.168.4/24; 192.168.5/24; }; // if .4 or .5, prefer that net
}; };
```

The following example will give reasonable behaviour for the local host and hosts on directly connected networks. It is similar to the behavior of the address sort in BIND 4.9.x. Responses sent to queries from the local host will favor any of the directly connected networks. Responses sent to queries from any other hosts on a directly connected network will prefer addresses on that same network. Responses to other queries will not be sorted.

```
sortlist {
      { localhost; localnets; };
      { localnets; }; };
```

When multiple records are returned in an answer it may be useful to configure the order the records are placed into the response. For example the records for a zone might be configured to always be returned in the order they are defined in the zone file. Or perhaps a random shuffle of the records as they are returned is wanted. The rrset-order statement permits configuration of the ordering made of the records in a multiple record response. The default, if no ordering is defined, is a cyclic ordering (round robin).

An is defined as follows:

[*class class_name*][*type type_name*][*name* "FQDN"] *order* ordering

If no class is specified, the default is If no is specified, the default is If no name is specified, the default is "*".

The legal values for are:

Records are returned in the order they are defined in the zone file.
Records are returned in some random order.
Records are returned in a round-robin order.

For example:

```
rrset-order {
   class IN type A name "rc.vix.com" order random;
   order cyclic;
};
```

will cause any responses for type A records in class IN that have "rc.vix.com" as a suffix, to always be returned in random order. All other records are returned in cyclic order.

If multiple statements appear, they are not combined--the last one applies.

If no statement is specified, a default one of:

```
rrset-order { class ANY type ANY name "*" order cyclic ; };
```

is used.

Sets the number of seconds to cache a lame server indication. 0 disables caching. Default is 600 (10 minutes). Maximum value is 1800 (30 minutes) To reduce network traffic and increase performance the server store negative answers. is used to set a maximum retention time for these answers in the server is seconds. The default is 10800 seconds (3 hours). cannot exceed the maximum retention time for ordinary (positive) answers (7 days) and will be silently truncated to 7 days if set to a value which is greater that 7 days. The minimum number of root servers that is required for a request for the root servers to be accepted. Default is 2.

zone *domain_name* [(in | hs | hesiod | chaos)] {
  type master;

```
  file path_name;
  [ check-names ( warn | fail | ignore ); ]
  [ allow-update { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ dialup yes_or_no; ]
  [ notify yes_or_no; ]
  [ also-notify { ip_addr; [ ip_addr; ... ] };
  [ pubkey number number number string; ] };

zone domain_name [ ( in | hs | hesiod | chaos ) ] {
  type ( slave | stub );
  [ file path_name; ]
  masters [ port ip_port ] { ip_addr; [ ip_addr; ... ] };
  [ check-names ( warn | fail | ignore ); ]
  [ allow-update { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ transfer-source ip_addr; ]
  [ max-transfer-time-in number; ]
  [ notify yes_or_no; ]
  [ also-notify { ip_addr; [ ip_addr; ... ] };
  [ pubkey number number number string; ] };

zone domain_name [ ( in | hs | hesiod | chaos ) ] {
  type forward;
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr ; [ ip_addr ; ... ] ] }; ]
  [ check-names ( warn | fail | ignore ); ] };

zone "." [ ( in | hs | hesiod | chaos ) ] {
  type hint;
  file path_name;
  [ check-names ( warn | fail | ignore ); ] };
```

The statement is used to define how information about particular DNS zones is managed by the server. There are five different zone types.

The server has a master copy of the data for the zone and will be able to provide authoritative answers for it.

A zone is a replica of a master zone. The list specifies one or more IP addresses that the slave contacts to update its copy of the zone. If a is specified then checks to see if the zone is current and zone transfers will be done to the port given. If is specified, then the replica will be written to the named file. Use of the clause is highly recommended, since it often speeds server startup and eliminates a needless waste of bandwidth.

A zone is like a slave zone, except that it replicates only the NS records of a master zone instead of the entire zone.

A zone is used to direct all queries in it to other servers, as described in section. The specification of options in such a zone will override any global options declared in the statement.

If either no clause is present in the zone or an empty list for is given, then no forwarding will be done for the zone, cancelling the effects of any in the statement. Thus if you want to use this type of zone to change only the behavior of the global option, and not the servers used, then you also need to respecify the global forwarders.

The initial set of root nameservers is specified using a zone. When the server starts up, it uses the root hints to find a root nameserver and get the most recent list of root nameservers.

Note: previous releases of BIND used the term for a master zone, for a slave zone, and for a hint zone.

The zone's name may optionally be followed by a class. If a class is not specified, class (for "internet"), is assumed. This is correct for the vast majority of cases.

The class is for an information service from MIT's Project Athena. It is used to share information about various systems databases, such as users, groups, printers and so on. More information can be found at ftp://athenadist.mit.edu/pub/ATHENA/usenix/athena_changes.PS. The keyword is a synonym for

Another MIT development was CHAOSnet, a LAN protocol created in the mid-1970s. It is still sometimes seen on LISP stations and other hardware in the AI community, and zone data for it can be specified with the class.

See the subsection on in

See the description of in the subsection of

Specifies which hosts are allowed to submit Dynamic DNS updates to the server. The default is to deny updates from all hosts.

See the description of in the subsection of

determines which local address will be bound to the TCP connection used to fetch this zone. If not set, it defaults to a system controlled value which will usually be the address of the interface ‘‘closest to’’ the remote end. This address must appear in the remote end's option for this zone if one is specified.

See the description of in the subsection of

See the description of in the subsection of

See the description of in the subsection of the

is only meaningful if is active for this zone. The set of machines that will receive a DNS NOTIFY message for this zone is made up of all the listed nameservers for the zone (other than the primary master) plus any IP addresses specified with is not meaningful for zones. The default is the empty list.

is only meaningful if the zone has a list. The value causes the lookup to fail after trying the and getting no answer, while would allow a normal lookup to be tried.

The option in a zone is used to override the list of global forwarders. If it is not specified in a zone of type forwarding is done for the zone; the global options are not used.

The DNSSEC flags, protocol, and algorithm are specified, as well as a base-64 encoded string representing the key.

acl *name* {
  address_match_list };

The statement creates a named address match list. It gets its name from a primary use of address match lists: Access Control Lists (ACLs).

Note that an address match list's name must be defined with before it can be used elsewhere; no forward references are allowed.

The following ACLs are built-in:

Allows all hosts. Denies all hosts. Allows the IP addresses of all interfaces on the system. Allows any host on a network for which the system has an interface.

key *key_id* {
  algorithm *algorithm_id*;
  secret *secret_string*; };

The statement defines a key ID which can be used in a statement to associate a method of authentication with a particular name server that is more rigorous than simple IP address matching. A key ID must be created with the statement before it can be used in a definition or an address match list.

The is a string that specifies a security/authentication algorithm. is the secret to be used by the algorithm, and is treated as a base-64 encoded string. It should go without saying, but probably can't, that if you have in your then it should not be readable by anyone but the superuser.

trusted-keys {
  [*domain_name flags protocol algorithm key*;] };

The statement is for use with DNSSEC-style security, originally specified in RFC 2065. DNSSEC is meant to provide three distinct services: key distribution, data origin authentication, and transaction and request authentication. A complete description of DNSSEC and its use is beyond the scope of this document, and readers interested in more information should start with RFC 2065 and then continue with the Internet Drafts available at http://www.ietf.org/ids.by.wg/dnssec.html.

Each trusted key is associated with a domain name. Its attributes are the non-negative integral and as well as a base-64 encoded string representing the

Any number of trusted keys can be specified.

server *ip_addr* {
  [bogus *yes_or_no*;]
  [transfers *number*;]
  [transfer-format (one-answer | many-answers);]
  [keys { *key_id* [*key_id* ...] };] };

The server statement defines the characteristics to be associated with a remote name server.

If you discover that a server is giving out bad data, marking it as will prevent further queries to it. The default value of is

The server supports two zone transfer methods. The first, uses one DNS message per resource record transferred. packs as many resource records as possible into a message. is more efficient, but is only known to be understood by BIND 8.1 and patched versions of BIND 4.9.5. You can specify which method to use for a server with the option. If is not specified, the specified by the statement will be used.

The will be used in a future release of the server to limit the number of concurrent in-bound zone transfers from the specified server. It is checked for syntax but is otherwise ignored.

The clause is used to identify a defined by the statement, to be used for transaction security when talking to the remote server. The statememnt must come before the statement that references it.

The statement is intended for future use by the server. It is checked for syntax but is otherwise ignored.

controls {
  [inet *ip_addr*
    port *ip_port*
    allow { *address_match_list*; };]
  [unix *path_name*
    perm *number*
    owner *number*
    group *number*;] };

The statement declares control channels to be used by system administrators to affect the operation of the local name server. These control channels are used by the utility to send commands to and retrieve non-DNS results from a name server.

A control channel is a FIFO in the file system, and access to it is controlled by normal file system permissions. It is created by with the specified file mode bits (see user and group owner. Note that, unlike the mode bits specified for will normally have a leading so the number is interpreted as octal. Also note that the user and group ownership specified as and must be given as numbers, not names. It is recommended that the permissions be restricted to administrative personnel only, or else any user on the system might be able to manage the local name server.

An control channel is a TCP/IP socket accessible to the Internet, created at the specified on the specified Modern clients are capable of speaking directly to these sockets, and the control protocol is ARPAnet-style text. It is recommended that 127.0.0.1 be the only used, and this only if you trust all non-privileged users on the local host to manage your name server.

include *path_name*;

The statement inserts the specified file at the point that the statement is encountered. It cannot be used within another statement, though, so a line such as is not allowed.

Use to break the configuration up into easily-managed chunks. For example:

include "/etc/security/keys.bind"; include "/etc/acls.bind";

could be used at the top of a BIND configuration file in order to include any ACL or key information.

Be careful not to type ‘‘#include’’, like you would in a C program, because ‘‘#’’ is used to start a comment.

The simplest configuration file that is still realistically useful is one which simply defines a hint zone that has a full path to the root servers file. zone "." in {
    type hint;
    file "/var/named/root.cache"; };

Here’s a more typical real-world example.

```
/*
 * A simple BIND 8 configuration
 */

logging {        category lame-servers { null; };        category cname { null; }; };

options {        directory "/var/named"; };

controls {        inet * port 52 allow { any; };                // a bad idea        unix "/var/run/ndc"
perm 0600 owner 0 group 0;    // the default };

zone "isc.org" in {        type master;        file "master/isc.org"; };

zone "vix.com" in {        type slave;        file "slave/vix.com";        masters { 10.0.0.53; }; };

zone "0.0.127.in-addr.arpa" in {        type master;        file "master/127.0.0"; };

zone "." in {        type hint;        file "root.cache"; };
```

The BIND 8 configuration file.