BIND 9 Administrator Reference Manual

August 1, 2006

Chapter 1

Introduction

The Internet Domain Name System (DNS) consists of the syntax to specify the names of entities in the Internet in a hierarchical manner, the rules used for delegating authority over names, and the system implementation that actually maps names to Internet addresses. DNS data is maintained in a group of distributed hierarchical databases.

1.1 Scope of Document

The Berkeley Internet Name Domain (BIND) implements a domain name server for a number of operating systems. This document provides basic information about the installation and care of the Internet Software Consortium (ISC) BIND version 9 software package for system administrators.

This version of the manual corresponds to BIND version 9.3.

1.2 Organization of This Document

In this document, *Section 1* introduces the basic DNS and BIND concepts. *Section 2* describes resource requirements for running BIND in various environments. Information in *Section 3* is *task-oriented* in its presentation and is organized functionally, to aid in the process of installing the BIND 9 software. The task-oriented section is followed by *Section 4*, which contains more advanced concepts that the system administrator may need for implementing certain options. *Section 5* describes the BIND 9 lightweight resolver. The contents of *Section 6* are organized as in a reference manual to aid in the ongoing maintenance of the software. *Section 7*addresses security considerations, and *Section 8* contains troubleshooting help. The main body of the document is followed by several *Appendices* which contain useful reference information, such as a *Bibliography* and historic information related to BIND and the Domain Name System.

1.3 Conventions Used in This Document

In this document, we use the following general typographic conventions:

To describe:	*We use the style:*
a pathname, filename, URL, hostname, mailing list name, or new term or concept	`Fixed width`
literal user input	**`Fixed Width Bold`**
program output	`Fixed Width`

The following conventions are used in descriptions of the BIND configuration file:

To describe:	*We use the style:*
keywords	`Fixed Width`
variables	`Fixed Width`
Optional input	[Text is enclosed in square brackets]

1.4 The Domain Name System (DNS)

The purpose of this document is to explain the installation and upkeep of the BIND software package, and we begin by reviewing the fundamentals of the Domain Name System (DNS) as they relate to BIND.

1.4.1 DNS Fundamentals

The Domain Name System (DNS) is the hierarchical, distributed database. It stores information for mapping Internet host names to IP addresses and vice versa, mail routing information, and other data used by Internet applications.

Clients look up information in the DNS by calling a *resolver* library, which sends queries to one or more *name servers* and interprets the responses. The BIND 9 software distribution contains a name server, **named**, and two resolver libraries, **liblwres** and **libbind**.

1.4.2 Domains and Domain Names

The data stored in the DNS is identified by *domain names* that are organized as a tree according to organizational or administrative boundaries. Each node of the tree, called a *domain*, is given a label. The domain name of the node is the concatenation of all the labels on the path from the node to the *root* node. This is represented in written form as a string of labels listed from right to left and separated by dots. A label need only be unique within its parent domain.

For example, a domain name for a host at the company *Example, Inc.* could be `mail.example.com`, where `com` is the top level domain to which `ourhost.example.com` belongs, `example` is a subdomain of `com`, and `ourhost` is the name of the host.

For administrative purposes, the name space is partitioned into areas called *zones*, each starting at a node and extending down to the leaf nodes or to nodes where other zones start. The data for each zone is stored in a *name server*, which answers queries about the zone using the *DNS protocol*.

The data associated with each domain name is stored in the form of *resource records* (RRs). Some of the supported resource record types are described in Section 6.3.1.

For more detailed information about the design of the DNS and the DNS protocol, please refer to the standards documents listed in Section A.3.1.

1.4.3 Zones

To properly operate a name server, it is important to understand the difference between a *zone* and a *domain*.

As we stated previously, a zone is a point of delegation in the DNS tree. A zone consists of those contiguous parts of the domain tree for which a name server has complete information and over which it has authority. It contains all domain names from a certain point downward in the domain tree except those which are delegated to other zones. A delegation point is marked by one or more *NS records* in the parent zone, which should be matched by equivalent NS records at the root of the delegated zone.

For instance, consider the `example.com` domain which includes names such as `host.aaa.example.com` and `host.bbb.example.com` even though the `example.com` zone includes only delegations for the `aaa.example.com` and `bbb.example.com` zones. A zone can map exactly to a single domain, but could also include only part of a domain, the rest of which could be delegated to other name servers. Every name in the DNS tree is a *domain*, even if it is *terminal*, that is, has no *subdomains*. Every subdomain is a domain and every domain except the root is also a subdomain. The terminology is not intuitive and we suggest that you read RFCs 1033, 1034 and 1035 to gain a complete understanding of this difficult and subtle topic.

Though BIND is called a "domain name server", it deals primarily in terms of zones. The master and slave declarations in the `named.conf` file specify zones, not domains. When you ask some other site if it is willing to be a slave server for your *domain*, you are actually asking for slave service for some collection of zones.

1.4.4 Authoritative Name Servers

Each zone is served by at least one *authoritative name server*, which contains the complete data for the zone. To make the DNS tolerant of server and network failures, most zones have two or more authoritative servers.

Responses from authoritative servers have the "authoritative answer" (AA) bit set in the response packets. This makes them easy to identify when debugging DNS configurations using tools like **dig** (Section 3.3.1.1).

1.4.4.1 The Primary Master

The authoritative server where the master copy of the zone data is maintained is called the *primary master* server, or simply the *primary*. It loads the zone contents from some local file edited by humans or perhaps generated mechanically from some other local file which is edited by humans. This file is called the *zone file* or *master file*.

1.4.4.2 Slave Servers

The other authoritative servers, the *slave* servers (also known as *secondary* servers) load the zone contents from another server using a replication process known as a *zone transfer*. Typically the data are transferred directly from the primary master, but it is also possible to transfer it from another slave. In other words, a slave server may itself act as a master to a subordinate slave server.

1.4.4.3 Stealth Servers

Usually all of the zone's authoritative servers are listed in NS records in the parent zone. These NS records constitute a *delegation* of the zone from the parent. The authoritative servers are also listed in the zone file itself, at the *top level* or *apex* of the zone. You can list servers in the zone's top-level NS records that are not in the parent's NS delegation, but you cannot list servers in the parent's delegation that are not present at the zone's top level.

A *stealth server* is a server that is authoritative for a zone but is not listed in that zone's NS records. Stealth servers can be used for keeping a local copy of a zone to speed up access to the zone's records or to make sure that the zone is available even if all the "official" servers for the zone are inaccessible.

A configuration where the primary master server itself is a stealth server is often referred to as a "hidden primary" configuration. One use for this configuration is when the primary master is behind a firewall and therefore unable to communicate directly with the outside world.

1.4.5 Caching Name Servers

The resolver libraries provided by most operating systems are *stub resolvers*, meaning that they are not capable of performing the full DNS resolution process by themselves by talking directly to the authoritative servers. Instead, they rely on a local name server to perform the resolution on their behalf. Such a server is called a *recursive* name server; it performs *recursive lookups* for local clients.

To improve performance, recursive servers cache the results of the lookups they perform. Since the processes of recursion and caching are intimately connected, the terms *recursive server* and *caching server* are often used synonymously.

The length of time for which a record may be retained in the cache of a caching name server is controlled by the Time To Live (TTL) field associated with each resource record.

1.4.5.1 Forwarding

Even a caching name server does not necessarily perform the complete recursive lookup itself. Instead, it can *forward* some or all of the queries that it cannot satisfy from its cache to another caching name server, commonly referred to as a *forwarder*.

There may be one or more forwarders, and they are queried in turn until the list is exhausted or an answer is found. Forwarders are typically used when you do not wish all the servers at a given site to interact directly with the rest of the Internet servers. A typical scenario would involve a number of internal DNS servers and an Internet firewall. Servers unable to pass packets through the firewall would forward to the server that can do it, and that server would query the Internet DNS servers on the internal server's behalf. An added benefit of using the forwarding feature is that the central machine develops a much more complete cache of information that all the clients can take advantage of.

1.4.6 Name Servers in Multiple Roles

The BIND name server can simultaneously act as a master for some zones, a slave for other zones, and as a caching (recursive) server for a set of local clients.

However, since the functions of authoritative name service and caching/recursive name service are logically separate, it is often advantageous to run them on separate server machines. A server that only provides authoritative name service (an *authoritative-only* server) can run with recursion disabled, improving reliability and security. A server that is not authoritative for any zones and only provides recursive service to local clients (a *caching-only* server) does not need to be reachable from the Internet at large and can be placed inside a firewall.

Chapter 2

BIND Resource Requirements

2.1 Hardware requirements

DNS hardware requirements have traditionally been quite modest. For many installations, servers that have been pensioned off from active duty have performed admirably as DNS servers.

The DNSSEC and IPv6 features of BIND 9 may prove to be quite CPU intensive however, so organizations that make heavy use of these features may wish to consider larger systems for these applications. BIND 9 is fully multithreaded, allowing full utilization of multiprocessor systems for installations that need it.

2.2 CPU Requirements

CPU requirements for BIND 9 range from i486-class machines for serving of static zones without caching, to enterprise-class machines if you intend to process many dynamic updates and DNSSEC signed zones, serving many thousands of queries per second.

2.3 Memory Requirements

The memory of the server has to be large enough to fit the cache and zones loaded off disk. The **max-cache-size** option can be used to limit the amount of memory used by the cache, at the expense of reducing cache hit rates and causing more DNS traffic. It is still good practice to have enough memory to load all zone and cache data into memory — unfortunately, the best way to determine this for a given installation is to watch the name server in operation. After a few weeks the server process should reach a relatively stable size where entries are expiring from the cache as fast as they are being inserted.

2.4 Name Server Intensive Environment Issues

For name server intensive environments, there are two alternative configurations that may be used. The first is where clients and any second-level internal name servers query a main name server, which has enough memory to build a large cache. This approach minimizes the bandwidth used by external name lookups. The second alternative is to set up second-level internal name servers to make queries independently. In this configuration, none of the individual machines needs to have as much memory or CPU power as in the first alternative, but this has the disadvantage of making many more external queries, as none of the name servers share their cached data.

2.5 Supported Operating Systems

ISC BIND 9 compiles and runs on a large number of Unix-like operating system and on Windows NT / 2000. For an up-to-date list of supported systems, see the README file in the top level directory of the BIND 9 source distribution.

Chapter 3

Name Server Configuration

In this section we provide some suggested configurations along with guidelines for their use. We also address the topic of reasonable option setting.

3.1 Sample Configurations

3.1.1 A Caching-only Name Server

The following sample configuration is appropriate for a caching-only name server for use by clients internal to a corporation. All queries from outside clients are refused using the **allow-query** option. Alternatively, the same effect could be achieved using suitable firewall rules.

```
// Two corporate subnets we wish to allow queries from.
acl corpnets { 192.168.4.0/24; 192.168.7.0/24; };
options {
     directory "/etc/namedb";           // Working directory
     allow-query { corpnets; };
};
// Provide a reverse mapping for the loopback address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
     type master;
     file "localhost.rev";
     notify no;
};
```

3.1.2 An Authoritative-only Name Server

This sample configuration is for an authoritative-only server that is the master server for "`example.com`" and a slave for the subdomain "`eng.example.com`".

```
options {
     directory "/etc/namedb";           // Working directory
     allow-query { any; };              // This is the default
     recursion no;                      // Do not provide recursive service
};

// Provide a reverse mapping for the loopback address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
```

```
     type master;
     file "localhost.rev";
     notify no;
};
// We are the master server for example.com
zone "example.com" {
     type master;
     file "example.com.db";
     // IP addresses of slave servers allowed to transfer example.com
     allow-transfer {
          192.168.4.14;
          192.168.5.53;
     };
};
// We are a slave server for eng.example.com
zone "eng.example.com" {
     type slave;
     file "eng.example.com.bk";
     // IP address of eng.example.com master server
     masters { 192.168.4.12; };
};
```

3.2 Load Balancing

A primitive form of load balancing can be achieved in the DNS by using multiple A records for one name.

For example, if you have three WWW servers with network addresses of 10.0.0.1, 10.0.0.2 and 10.0.0.3, a set of records such as the following means that clients will connect to each machine one third of the time:

Name	TTL	CLASS	TYPE	Resource Record (RR) Data
www	600	IN	A	10.0.0.1
	600	IN	A	10.0.0.2
	600	IN	A	10.0.0.3

When a resolver queries for these records, BIND will rotate them and respond to the query with the records in a different order. In the example above, clients will randomly receive records in the order 1, 2, 3; 2, 3, 1; and 3, 1, 2. Most clients will use the first record returned and discard the rest.

For more detail on ordering responses, check the **rrset-order** substatement in the **options** statement, see RRset Ordering. This substatement is not supported in BIND 9, and only the ordering scheme described above is available.

3.3 Name Server Operations

3.3.1 Tools for Use With the Name Server Daemon

There are several indispensable diagnostic, administrative and monitoring tools available to the system administrator for controlling and debugging the name server daemon. We describe several in this section

3.3.1.1 Diagnostic Tools

The **dig**, **host**, and **nslookup** programs are all command line tools for manually querying name servers. They differ in style and output format.

dig The domain information groper (**dig**) is the most versatile and complete of these lookup tools. It has two modes: simple interactive mode for a single query, and batch mode which executes a query for each in a list of several query lines. All query options are accessible from the command line.

Usage

```
dig [@server] domain [query-type] [query-class] [+query-option]
    [-dig-option] [%comment]
```

The usual simple use of dig will take the form

dig @server domain query-type query-class

For more information and a list of available commands and options, see the **dig** man page.

host The **host** utility emphasizes simplicity and ease of use. By default, it converts between host names and Internet addresses, but its functionality can be extended with the use of options.

Usage

```
host [-aCdlrTwv] [-c class] [-N ndots] [-t type] [-W timeout] [-R
    retries] hostname [server]
```

For more information and a list of available commands and options, see the **host** man page.

nslookup **nslookup** has two modes: interactive and non-interactive. Interactive mode allows the user to query name servers for information about various hosts and domains or to print a list of hosts in a domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

Usage

```
nslookup [-option...]  [host-to-find | - [server]]
```

Interactive mode is entered when no arguments are given (the default name server will be used) or when the first argument is a hyphen ('-') and the second argument is the host name or Internet address of a name server.

Non-interactive mode is used when the name or Internet address of the host to be looked up is given as the first argument. The optional second argument specifies the host name or address of a name server.

Due to its arcane user interface and frequently inconsistent behavior, we do not recommend the use of **nslookup**. Use **dig** instead.

3.3.1.2 Administrative Tools

Administrative tools play an integral part in the management of a server.

named-checkconf The **named-checkconf** program checks the syntax of a `named.conf` file.

Usage

```
named-checkconf [-jvz] [-t directory] [filename]
```

named-checkzone The **named-checkzone** program checks a master file for syntax and consistency.

Usage

```
named-checkzone [-djqvD] [-c class] [-o output] [-t directory] [-w
    directory] [-k (ignore|warn|fail)] [-n (ignore|warn|fail)] zone
    [filename]
```

rndc The remote name daemon control (**rndc**) program allows the system administrator to control the operation of a name server. If you run **rndc** without any options it will display a usage message as follows:

Usage
```
rndc [-c config] [-s server] [-p port] [-y key] command [command...]
```

The **command** is one of the following:

reload Reload configuration file and zones.

reload *zone* [*class* [*view*]] Reload the given zone.

refresh *zone* [*class* [*view*]] Schedule zone maintenance for the given zone.

retransfer *zone* [*class* [*view*]] Retransfer the given zone from the master.

freeze [*zone* [*class* [*view*]]] Suspend updates to a dynamic zone. If no zone is specified, then all zones are suspended. This allows manual edits to be made to a zone normally updated by dynamic update. It also causes changes in the journal file to be synced into the master and the journal file to be removed. All dynamic update attempts will be refused while the zone is frozen.

thaw [*zone* [*class* [*view*]]] Enable updates to a frozen dynamic zone. If no zone is specified, then all frozen zones are enabled. This causes the server to reload the zone from disk, and re-enables dynamic updates after the load has completed. After a zone is thawed, dynamic updates will no longer be refused.

reconfig Reload the configuration file and load new zones, but do not reload existing zone files even if they have changed. This is faster than a full **reload** when there is a large number of zones because it avoids the need to examine the modification times of the zones files.

stats Write server statistics to the statistics file.

querylog Toggle query logging. Query logging can also be enabled by explicitly directing the **queries category** to a **channel** in the **logging** section of `named.conf`.

dumpdb [-all|-cache|-zone] [*view ...*] Dump the server's caches (default) and / or zones to the dump file for the specified views. If no view is specified, all views are dumped.

stop [-p] Stop the server, making sure any recent changes made through dynamic update or IXFR are first saved to the master files of the updated zones. If -p is specified named's process id is returned. This allows an external process to determine when named had completed stopping.

halt [-p] Stop the server immediately. Recent changes made through dynamic update or IXFR are not saved to the master files, but will be rolled forward from the journal files when the server is restarted. If -p is specified named's process id is returned. This allows an external process to determine when named had completed stopping.

`trace` Increment the servers debugging level by one.

`trace` ***`level`*** Sets the server's debugging level to an explicit value.

`notrace` Sets the server's debugging level to 0.

`flush` Flushes the server's cache.

`flushname` ***`name`*** Flushes the given name from the server's cache.

`status` Display status of the server. Note that the number of zones includes the internal **bind/CH** zone and the default **./IN** hint zone if there is not an explicit root zone configured.

`recursing` Dump the list of queries named is currently recursing on.

In BIND 9.2, **rndc** supports all the commands of the BIND 8 **ndc** utility except **ndc start** and **ndc restart**, which were also not supported in **ndc**'s channel mode.

A configuration file is required, since all communication with the server is authenticated with digital signatures that rely on a shared secret, and there is no way to provide that secret other than with a configuration file. The default location for the **rndc** configuration file is `/etc/rndc.conf`, but an alternate location can be specified with the `-c` option. If the configuration file is not found, **rndc** will also look in `/etc/rndc.key` (or whatever `sysconfdir` was defined when the BIND build was configured). The `rndc.key` file is generated by running **rndc-confgen -a** as described in Section 6.2.4.

The format of the configuration file is similar to that of `named.conf`, but limited to only four statements, the **options**, **key**, **server** and **include** statements. These statements are what associate the secret keys to the servers with which they are meant to be shared. The order of statements is not significant.

The **options** statement has three clauses: **default-server**, **default-key**, and **default-port**. **default-server** takes a host name or address argument and represents the server that will be contacted if no `-s` option is provided on the command line. **default-key** takes the name of a key as its argument, as defined by a **key** statement. **default-port** specifies the port to which **rndc** should connect if no port is given on the command line or in a **server** statement.

The **key** statement defines a key to be used by **rndc** when authenticating with **named**. Its syntax is identical to the **key** statement in named.conf. The keyword **`key`** is followed by a key name, which must be a valid domain name, though it need not actually be hierarchical; thus, a string like "**`rndc_key`**" is a valid name. The **key** statement has two clauses: **algorithm** and **secret**. While the configuration parser will accept any string as the argument to algorithm, currently only the string "**`hmac-md5`**" has any meaning. The secret is a base-64 encoded string.

The **server** statement associates a key defined using the **key** statement with a server. The keyword **`server`** is followed by a host name or address. The **server** statement has two clauses: **key** and **port**. The **key** clause specifies the name of the key to be used when communicating with this server, and the **port** clause can be used to specify the port **rndc** should connect to on the server.

A sample minimal configuration file is as follows:

```
key rndc_key {
     algorithm "hmac-md5";
     secret "c3Ryb25nIGVub3VnaCBmb3IgYSBtYW4gYnV0IG1hZGUgZm9yIGEgd29tYW4K";
};
options {
     default-server 127.0.0.1;
```

```
    default-key     rndc_key;
};
```

This file, if installed as `/etc/rndc.conf`, would allow the command:

`$`**`rndc reload`**

to connect to 127.0.0.1 port 953 and cause the name server to reload, if a name server on the local machine were running with following controls statements:

```
controls {
        inet 127.0.0.1 allow { localhost; } keys { rndc_key; };
};
```

and it had an identical key statement for `rndc_key`.

Running the **rndc-confgen** program will conveniently create a `rndc.conf` file for you, and also display the corresponding **controls** statement that you need to add to `named.conf`. Alternatively, you can run **rndc-confgen -a** to set up a `rndc.key` file and not modify `named.conf` at all.

3.3.2 Signals

Certain UNIX signals cause the name server to take specific actions, as described in the following table. These signals can be sent using the **kill** command.

SIGHUP	Causes the server to read `named.conf` and reload the database.
SIGTERM	Causes the server to clean up and exit.
SIGINT	Causes the server to clean up and exit.

Chapter 4

Advanced DNS Features

4.1 Notify

DNS NOTIFY is a mechanism that allows master servers to notify their slave servers of changes to a zone's data. In response to a **NOTIFY** from a master server, the slave will check to see that its version of the zone is the current version and, if not, initiate a zone transfer.

DNS For more information about **NOTIFY**, see the description of the **notify** option in Section 6.2.16.1 and the description of the zone option **also-notify** in Section 6.2.16.7. The **NOTIFY** protocol is specified in RFC 1996.

4.2 Dynamic Update

Dynamic Update is a method for adding, replacing or deleting records in a master server by sending it a special form of DNS messages. The format and meaning of these messages is specified in RFC 2136.

Dynamic update is enabled on a zone-by-zone basis, by including an **allow-update** or **update-policy** clause in the **zone** statement.

Updating of secure zones (zones using DNSSEC) follows RFC 3007: RRSIG and NSEC records affected by updates are automatically regenerated by the server using an online zone key. Update authorization is based on transaction signatures and an explicit server policy.

4.2.1 The journal file

All changes made to a zone using dynamic update are stored in the zone's journal file. This file is automatically created by the server when the first dynamic update takes place. The name of the journal file is formed by appending the extension `.jnl` to the name of the corresponding zone file. The journal file is in a binary format and should not be edited manually.

The server will also occasionally write ("dump") the complete contents of the updated zone to its zone file. This is not done immediately after each dynamic update, because that would be too slow when a large zone is updated frequently. Instead, the dump is delayed by up to 15 minutes, allowing additional updates to take place.

When a server is restarted after a shutdown or crash, it will replay the journal file to incorporate into the zone any updates that took place after the last zone dump.

Changes that result from incoming incremental zone transfers are also journalled in a similar way.

The zone files of dynamic zones cannot normally be edited by hand because they are not guaranteed to contain the most recent dynamic changes — those are only in the journal file. The only way to ensure that the zone file of a dynamic zone is up to date is to run **rndc stop**.

If you have to make changes to a dynamic zone manually, the following procedure will work: Disable dynamic updates to the zone using **rndc freeze *zone***. This will also remove the zone's `.jnl` file and update the master file. Edit the zone file. Run **rndc unfreeze *zone*** to reload the changed zone and re-enable dynamic updates.

4.3 Incremental Zone Transfers (IXFR)

The incremental zone transfer (IXFR) protocol is a way for slave servers to transfer only changed data, instead of having to transfer the entire zone. The IXFR protocol is specified in RFC 1995. See [Proposed Standards].

When acting as a master, BIND 9 supports IXFR for those zones where the necessary change history information is available. These include master zones maintained by dynamic update and slave zones whose data was obtained by IXFR. For manually maintained master zones, and for slave zones obtained by performing a full zone transfer (AXFR), IXFR is supported only if the option **ixfr-from-differences** is set to **`yes`**.

When acting as a slave, BIND 9 will attempt to use IXFR unless it is explicitly disabled. For more information about disabling IXFR, see the description of the **request-ixfr** clause of the **server** statement.

4.4 Split DNS

Setting up different views, or visibility, of the DNS space to internal and external resolvers is usually referred to as a *Split DNS* setup. There are several reasons an organization would want to set up its DNS this way.

One common reason for setting up a DNS system this way is to hide "internal" DNS information from "external" clients on the Internet. There is some debate as to whether or not this is actually useful. Internal DNS information leaks out in many ways (via email headers, for example) and most savvy "attackers" can find the information they need using other means.

Another common reason for setting up a Split DNS system is to allow internal networks that are behind filters or in RFC 1918 space (reserved IP space, as documented in RFC 1918) to resolve DNS on the Internet. Split DNS can also be used to allow mail from outside back in to the internal network.

Here is an example of a split DNS setup:

Let's say a company named *Example, Inc.* (`example.com`) has several corporate sites that have an internal network with reserved Internet Protocol (IP) space and an external demilitarized zone (DMZ), or "outside" section of a network, that is available to the public.

Example, Inc. wants its internal clients to be able to resolve external hostnames and to exchange mail with people on the outside. The company also wants its internal resolvers to have access to certain internal-only zones that are not available at all outside of the internal network.

In order to accomplish this, the company will set up two sets of name servers. One set will be on the inside network (in the reserved IP space) and the other set will be on bastion hosts, which are "proxy" hosts that can talk to both sides of its network, in the DMZ.

The internal servers will be configured to forward all queries, except queries for `site1.internal`, `site2.internal`, `site1.example.com`, and `site2.example.com`, to the servers in the DMZ. These internal servers will have complete sets of information for `site1.example.com`, `site2.example.com`,`site1.internal`, and `site2.internal`.

To protect the `site1.internal` and `site2.internal` domains, the internal name servers must be configured to disallow all queries to these domains from any external hosts, including the bastion hosts.

The external servers, which are on the bastion hosts, will be configured to serve the "public" version of the `site1` and `site2.example.com` zones. This could include things such as the host records for public servers (`www.example.com` and `ftp.example.com`), and mail exchange (MX) records (`a.mx.example.com` and `b.mx.example.com`).

In addition, the public `site1` and `site2.example.com` zones should have special MX records that contain wildcard ('*') records pointing to the bastion hosts. This is needed because external mail servers do not have any other way of looking up how to deliver mail to those internal hosts. With the wildcard records, the mail will be delivered to the bastion host, which can then forward it on to internal hosts.

Here's an example of a wildcard MX record:

```
*   IN MX 10 external1.example.com.
```

Now that they accept mail on behalf of anything in the internal network, the bastion hosts will need to know how to deliver mail to internal hosts. In order for this to work properly, the resolvers on the bastion hosts will need to be configured to point to the internal name servers for DNS resolution.

Queries for internal hostnames will be answered by the internal servers, and queries for external hostnames will be forwarded back out to the DNS servers on the bastion hosts.

In order for all this to work properly, internal clients will need to be configured to query *only* the internal name servers for DNS queries. This could also be enforced via selective filtering on the network.

If everything has been set properly, *Example, Inc.*'s internal clients will now be able to:

- Look up any hostnames in the `site1` and `site2.example.com` zones.
- Look up any hostnames in the `site1.internal` and `site2.internal` domains.
- Look up any hostnames on the Internet.
- Exchange mail with both internal AND external people.

Hosts on the Internet will be able to:

- Look up any hostnames in the `site1` and `site2.example.com` zones.
- Exchange mail with anyone in the `site1` and `site2.example.com` zones.

Here is an example configuration for the setup we just described above. Note that this is only configuration information; for information on how to configure your zone files, see Section 3.1.

Internal DNS server config:

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };

acl externals { bastion-ips-go-here; };

options {
    ...
    ...
    forward only;
    forwarders {                                    // forward to external servers
        bastion-ips-go-here;
    };
    allow-transfer { none; };                       // sample allow-transfer (no one)
    allow-query { internals; externals; };          // restrict query access
    allow-recursion { internals; };                 // restrict recursion
    ...
    ...
};

zone "site1.example.com" {                          // sample master zone
  type master;
  file "m/site1.example.com";
  forwarders { };                                   // do normal iterative
                                                    // resolution (do not forward)
  allow-query { internals; externals; };
  allow-transfer { internals; };
```

```
};

zone "site2.example.com" {                      // sample slave zone
  type slave;
  file "s/site2.example.com";
  masters { 172.16.72.3; };
  forwarders { };
  allow-query { internals; externals; };
  allow-transfer { internals; };
};

zone "site1.internal" {
  type master;
  file "m/site1.internal";
  forwarders { };
  allow-query { internals; };
  allow-transfer { internals; }
};

zone "site2.internal" {
  type slave;
  file "s/site2.internal";
  masters { 172.16.72.3; };
  forwarders { };
  allow-query { internals };
  allow-transfer { internals; }
};
```

External (bastion host) DNS server config:

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };

acl externals { bastion-ips-go-here; };

options {
  ...
  ...
  allow-transfer { none; };                     // sample allow-transfer (no one)
  allow-query { internals; externals; };        // restrict query access
  allow-recursion { internals; externals; };    // restrict recursion
  ...
  ...
};

zone "site1.example.com" {                      // sample slave zone
  type master;
  file "m/site1.foo.com";
  allow-query { any; };
  allow-transfer { internals; externals; };
};

zone "site2.example.com" {
  type slave;
  file "s/site2.foo.com";
  masters { another_bastion_host_maybe; };
  allow-query { any; };
  allow-transfer { internals; externals; }
};
```

In the `resolv.conf` (or equivalent) on the bastion host(s):

```
search ...
nameserver 172.16.72.2
nameserver 172.16.72.3
nameserver 172.16.72.4
```

4.5 TSIG

This is a short guide to setting up Transaction SIGnatures (TSIG) based transaction security in BIND. It describes changes to the configuration file as well as what changes are required for different features, including the process of creating transaction keys and using transaction signatures with BIND.

BIND primarily supports TSIG for server to server communication. This includes zone transfer, notify, and recursive query messages. Resolvers based on newer versions of BIND 8 have limited support for TSIG.

TSIG might be most useful for dynamic update. A primary server for a dynamic zone should use access control to control updates, but IP-based access control is insufficient. The cryptographic access control provided by TSIG is far superior. The **nsupdate** program supports TSIG via the `-k` and `-y` command line options.

4.5.1 Generate Shared Keys for Each Pair of Hosts

A shared secret is generated to be shared between *host1* and *host2*. An arbitrary key name is chosen: "host1-host2.". The key name must be the same on both hosts.

4.5.1.1 Automatic Generation

The following command will generate a 128-bit (16 byte) HMAC-MD5 key as described above. Longer keys are better, but shorter keys are easier to read. Note that the maximum key length is 512 bits; keys longer than that will be digested with MD5 to produce a 128-bit key.

```
dnssec-keygen -a hmac-md5 -b 128 -n HOST host1-host2.
```

The key is in the file `Khost1-host2.+157+00000.private`. Nothing directly uses this file, but the base-64 encoded string following "`Key:`" can be extracted from the file and used as a shared secret:

```
Key: La/E5CjG9O+os1jq0a2jdA==
```

The string "`La/E5CjG9O+os1jq0a2jdA==`" can be used as the shared secret.

4.5.1.2 Manual Generation

The shared secret is simply a random sequence of bits, encoded in base-64. Most ASCII strings are valid base-64 strings (assuming the length is a multiple of 4 and only valid characters are used), so the shared secret can be manually generated.

Also, a known string can be run through **mmencode** or a similar program to generate base-64 encoded data.

4.5.2 Copying the Shared Secret to Both Machines

This is beyond the scope of DNS. A secure transport mechanism should be used. This could be secure FTP, ssh, telephone, etc.

4.5.3 Informing the Servers of the Key's Existence

Imagine *host1* and *host 2* are both servers. The following is added to each server's `named.conf` file:

```
key host1-host2. {
  algorithm hmac-md5;
  secret "La/E5CjG9O+os1jq0a2jdA==";
};
```

The algorithm, hmac-md5, is the only one supported by BIND. The secret is the one generated above. Since this is a secret, it is recommended that either `named.conf` be non-world readable, or the key directive be added to a non-world readable file that is included by `named.conf`.

At this point, the key is recognized. This means that if the server receives a message signed by this key, it can verify the signature. If the signature is successfully verified, the response is signed by the same key.

4.5.4 Instructing the Server to Use the Key

Since keys are shared between two hosts only, the server must be told when keys are to be used. The following is added to the `named.conf` file for *host1*, if the IP address of *host2* is 10.1.2.3:

```
server 10.1.2.3 {
  keys { host1-host2. ;};
};
```

Multiple keys may be present, but only the first is used. This directive does not contain any secrets, so it may be in a world-readable file.

If *host1* sends a message that is a request to that address, the message will be signed with the specified key. *host1* will expect any responses to signed messages to be signed with the same key.

A similar statement must be present in *host2*'s configuration file (with *host1*'s address) for *host2* to sign request messages to *host1*.

4.5.5 TSIG Key Based Access Control

BIND allows IP addresses and ranges to be specified in ACL definitions and **allow-**{ **query** | **transfer** | **update** } directives. This has been extended to allow TSIG keys also. The above key would be denoted **key host1-host2.**

An example of an allow-update directive would be:

```
allow-update { key host1-host2. ;};
```

This allows dynamic updates to succeed only if the request was signed by a key named "**host1-host2.**".

You may want to read about the more powerful **update-policy** statement in Section 6.2.24.4.

4.5.6 Errors

The processing of TSIG signed messages can result in several errors. If a signed message is sent to a non-TSIG aware server, a FORMERR (format error) will be returned, since the server will not understand the record. This is a result of misconfiguration, since the server must be explicitly configured to send a TSIG signed message to a specific server.

If a TSIG aware server receives a message signed by an unknown key, the response will be unsigned with the TSIG extended error code set to BADKEY. If a TSIG aware server receives a message with a signature that does not validate, the response will be unsigned with the TSIG extended error code set to BADSIG. If a TSIG aware server receives a message with a time outside of the allowed range, the response will be signed with the TSIG extended error code set to BADTIME, and the time values will be adjusted so that the response can be successfully verified. In any of these cases, the message's rcode is set to NOTAUTH (not authenticated).

4.6 TKEY

TKEY is a mechanism for automatically generating a shared secret between two hosts. There are several "modes" of **TKEY** that specify how the key is generated or assigned. BIND 9 implements only one of these modes, the Diffie-Hellman key exchange. Both hosts are required to have a Diffie-Hellman KEY record (although this record is not required to be present in a zone). The **TKEY** process must use signed messages, signed either by TSIG or SIG(0). The result of **TKEY** is a shared secret that can be used to sign messages with TSIG. **TKEY** can also be used to delete shared secrets that it had previously generated.

The **TKEY** process is initiated by a client or server by sending a signed **TKEY** query (including any appropriate KEYs) to a TKEY-aware server. The server response, if it indicates success, will contain a **TKEY** record and any appropriate keys. After this exchange, both participants have enough information to determine the shared secret; the exact process depends on the **TKEY** mode. When using the Diffie-Hellman **TKEY** mode, Diffie-Hellman keys are exchanged, and the shared secret is derived by both participants.

4.7 SIG(0)

BIND 9 partially supports DNSSEC SIG(0) transaction signatures as specified in RFC 2535 and RFC2931. SIG(0) uses public/private keys to authenticate messages. Access control is performed in the same manner as TSIG keys; privileges can be granted or denied based on the key name.

When a SIG(0) signed message is received, it will only be verified if the key is known and trusted by the server; the server will not attempt to locate and / or validate the key.

SIG(0) signing of multiple-message TCP streams is not supported.

The only tool shipped with BIND 9 that generates SIG(0) signed messages is **nsupdate**.

4.8 DNSSEC

Cryptographic authentication of DNS information is possible through the DNS Security (*DNSSEC-bis*) extensions, defined in RFC 4033, RFC4034 and RFC4035. This section describes the creation and use of DNSSEC signed zones.

In order to set up a DNSSEC secure zone, there are a series of steps which must be followed. BIND 9 ships with several tools that are used in this process, which are explained in more detail below. In all cases, the `-h` option prints a full list of parameters. Note that the DNSSEC tools require the keyset files to be in the working directory or the directory specified by the `-h` option, and that the tools shipped with BIND 9.2.x and earlier are not compatible with the current ones.

There must also be communication with the administrators of the parent and/or child zone to transmit keys. A zone's security status must be indicated by the parent zone for a DNSSEC capable resolver to trust its data. This is done through the presence or absence of a `DS` record at the delegation point.

For other servers to trust data in this zone, they must either be statically configured with this zone's zone key or the zone key of another zone above this one in the DNS tree.

4.8.1 Generating Keys

The **dnssec-keygen** program is used to generate keys.

A secure zone must contain one or more zone keys. The zone keys will sign all other records in the zone, as well as the zone keys of any secure delegated zones. Zone keys must have the same name as the zone, a name type of **ZONE**, and must be usable for authentication. It is recommended that zone keys use a cryptographic algorithm designated as "mandatory to implement" by the IETF; currently the only one is RSASHA1.

The following command will generate a 768-bit RSASHA1 key for the `child.example` zone:

```
dnssec-keygen -a RSASHA1 -b 768 -n ZONE child.example.
```

Two output files will be produced: `Kchild.example.+005+12345.key` and `Kchild.example.+005+12345.private` (where 12345 is an example of a key tag). The key file names contain the key name (`child.example.`), algorithm (3 is DSA, 1 is RSAMD5, 5 is RSASHA1, etc.), and the key tag (12345 in this case). The private key (in the `.private` file) is used to generate signatures, and the public key (in the `.key` file) is used for signature verification.

To generate another key with the same properties (but with a different key tag), repeat the above command.

The public keys should be inserted into the zone file by including the `.key` files using **$INCLUDE** statements.

4.8.2 Signing the Zone

The **dnssec-signzone** program is used to sign a zone.

Any `keyset` files corresponding to secure subzones should be present. The zone signer will generate `NSEC` and `RRSIG` records for the zone, as well as `DS` for the child zones if `'-d'` is specified. If `'-d'` is not specified, then DS RRsets for the secure child zones need to be added manually.

The following command signs the zone, assuming it is in a file called `zone.child.example`. By default, all zone keys which have an available private key are used to generate signatures.

```
dnssec-signzone -o child.example zone.child.example
```

One output file is produced: `zone.child.example.signed`. This file should be referenced by `named.conf` as the input file for the zone.

dnssec-signzone will also produce a keyset and dsset files and optionally a dlvset file. These are used to provide the parent zone administators with the `DNSKEYs` (or their corresponding `DS` records) that are the secure entry point to the zone.

4.8.3 Configuring Servers

To enable **named** to respond appropriately to DNS requests from DNSSEC aware clients, **dnssec-enable** must be set to yes.

To enable **named** to validate answers from other servers **dnssec-enable** and some **trusted-keys** must be configured into `named.conf`.

trusted-keys are copies of DNSKEY RRs for zones that are used to form the first link in the cryptographic chain of trust. All keys listed in **trusted-keys** (and corresponding zones) are deemed to exist and only the listed keys will be used to validated the DNSKEY RRset that they are from.

trusted-keys are described in more detail later in this document.

Unlike BIND 8, BIND 9 does not verify signatures on load, so zone keys for authoritative zones do not need to be specified in the configuration file.

After DNSSEC gets established, a typical DNSSEC configuration will look something like the following. It has a one or more public keys for the root. This allows answers from outside the organization to be validated. It will also have several keys for parts of the namespace the organization controls. These are here to ensure that named is immune to compromises in the DNSSEC components of the security of parent zones.

```
trusted-keys {

   /* Root Key */
"." 257 3 3 "BNY4wrWM1nCfJ+CXd0rVXyYmobt7sEEfK3clRbGaTwSJxrGkxJWoZu6I7PzJu/
        E9gx4UC1zGAHlXKdE4zYIpRhaBKnvcC2U9mZhkdUpd1Vso/HAdjNe8LmMlnzY3
        zy2Xy4klWOADTPzSv9eamj8V18PHGjBLaVtYvk/ln5ZApjYghf+6fElrmLkdaz
        MQ2OCnACR817DF4BBa7UR/beDHyp5iWTXWSi6XmoJLbG9Scqc7l70KDqlvXR3M
        /lUUVRbkeg1IPJSidmK3ZyCllh4XSKbje/45SKucHgnwU5jefMtq66gKodQj+M
        iA21AfUVe7u99WzTLzY3qlxDhxYQQ20FQ97S+LKUTpQcq27R7AT3/V5hRQxScI
        Nqwcz4jYqZD2fQdgxbcDTClU0CRBdiieyLMNzXG3";

/* Key for our organization's forward zone */
example.com. 257 3 5 "AwEAAaxPMcR2x0HbQV4WeZB6oEDX+r0QM65KbhTjrW1ZaARmPhEZZe
                      3Y9ifgEuq7vZ/zGZUdEGNWy+JZzus0lUptwgjGwhUS1558Hb4JKUbb
                 OTcM8pwXlj0EiX3oDFVmjHO444gLkBO UKUf/mC7HvfwYH/Be22GnC
            lrinKJp1Og4ywzO9WglMk7jbfW33gUKvirTHr25GL7STQUzBb5Usxt
            8lgnyTUHs1t3JwCY5hKZ6CqFxmAVZP20igTixin/1LcrgX/KMEGd/b
            iuvF4qJCyduieHukuY3H4XMAcR+xia2 nIUPvm/oyWR8BW/hWdzOvn
            SCThlHf3xiYleDbt/o1OTQ09A0=";

/* Key for our reverse zone. */
2.0.192.IN-ADDRPA.NET. 257 3 5 "AQOnS4xn/IgOUpBPJ3bogzwcxOdNax071L18QqZnQQQA
                                VVr+iLhGTnNGp3HoWQLUIzKrJVZ3zggy3WwNT6kZo6c0
            tszYqbtvchmgQC8CzKojM/W16i6MG/ea fGU3siaOdS0
            yOI6BgPsw+YZdzlYMaIJGf4M4dyoKIhzdZyQ2bYQrjyQ
            4LB0lC7aOnsMyYKHHYeRv PxjIQXmdqgOJGq+vsevG06
                 zW+1xgYJh9rCIfnm1GX/KMgxLPG2vXTD/RnLX+D3T3UL
            7HJYHJhAZD5L59VvjSPsZJHeDCUyWYrvPZesZDIRvhDD
            52SKvbheeTJUm6EhkzytNN2SN96QRk8j/iI8ib";
};

options {
   ...
   dnssec-enable yes;
};
```

NOTE

None of the keys listed in this example are valid. In particular, the root key is not valid.

4.9 IPv6 Support in BIND 9

BIND 9 fully supports all currently defined forms of IPv6 name to address and address to name lookups. It will also use IPv6 addresses to make queries when running on an IPv6 capable system.

For forward lookups, BIND 9 supports only AAAA records. The use of A6 records is deprecated by RFC 3363, and the support for forward lookups in BIND 9 is removed accordingly. However, authoritative BIND 9 name servers still load zone files containing A6 records correctly, answer queries for A6 records, and accept zone transfer for a zone containing A6 records.

For IPv6 reverse lookups, BIND 9 supports the traditional "nibble" format used in the *ip6.arpa* domain, as well as the older, deprecated *ip6.int* domain. BIND 9 formerly supported the "binary label" (also known as "bitstring") format. The support of binary labels, however, is now completely removed according to the changes in RFC 3363. Any applications in BIND 9 do not understand the format any more, and will return an error if given. In particular, an authoritative BIND 9 name server rejects to load a zone file containing binary labels.

For an overview of the format and structure of IPv6 addresses, see Section A.2.1.

4.9.1 Address Lookups Using AAAA Records

The AAAA record is a parallel to the IPv4 A record. It specifies the entire address in a single record. For example,

```
$ORIGIN example.com.
host            3600    IN      AAAA    2001:db8::1
```

It is recommended that IPv4-in-IPv6 mapped addresses not be used. If a host has an IPv4 address, use an A record, not a AAAA, with `::ffff:192.168.42.1` as the address.

4.9.2 Address to Name Lookups Using Nibble Format

When looking up an address in nibble format, the address components are simply reversed, just as in IPv4, and `ip6.arpa.` is appended to the resulting name. For example, the following would provide reverse name lookup for a host with address `2001:db8::1`.

```
$ORIGIN 0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0   14400 IN      PTR     host.example.com.
```

Chapter 5

The BIND 9 Lightweight Resolver

5.1 The Lightweight Resolver Library

Traditionally applications have been linked with a stub resolver library that sends recursive DNS queries to a local caching name server.

IPv6 once introduced new complexity into the resolution process, such as following A6 chains and DNAME records, and simultaneous lookup of IPv4 and IPv6 addresses. Though most of the complexity was then removed, these are hard or impossible to implement in a traditional stub resolver.

Instead, BIND 9 provides resolution services to local clients using a combination of a lightweight resolver library and a resolver daemon process running on the local host. These communicate using a simple UDP-based protocol, the "lightweight resolver protocol" that is distinct from and simpler than the full DNS protocol.

5.2 Running a Resolver Daemon

To use the lightweight resolver interface, the system must run the resolver daemon **lwresd** or a local name server configured with a **lwres** statement.

By default, applications using the lightweight resolver library will make UDP requests to the IPv4 loopback address (127.0.0.1) on port 921. The address can be overridden by **lwserver** lines in `/etc/resolv.conf`.

The daemon currently only looks in the DNS, but in the future it may use other sources such as `/etc/hosts`, NIS, etc.

The **lwresd** daemon is essentially a caching-only name server that responds to requests using the lightweight resolver protocol rather than the DNS protocol. Because it needs to run on each host, it is designed to require no or minimal configuration. Unless configured otherwise, it uses the name servers listed on **nameserver** lines in `/etc/resolv.conf` as forwarders, but is also capable of doing the resolution autonomously if none are specified.

The **lwresd** daemon may also be configured with a `named.conf` style configuration file, in `/etc/lwresd.conf` by default. A name server may also be configured to act as a lightweight resolver daemon using the **lwres** statement in `named.conf`.

Chapter 6

BIND 9 Configuration Reference

BIND 9 configuration is broadly similar to BIND 8; however, there are a few new areas of configuration, such as views. BIND 8 configuration files should work with few alterations in BIND 9, although more complex configurations should be reviewed to check if they can be more efficiently implemented using the new features found in BIND 9.

BIND 4 configuration files can be converted to the new format using the shell script `contrib/named-bootconf/named-bootconf.sh`.

6.1 Configuration File Elements

Following is a list of elements used throughout the BIND configuration file documentation:

`acl_name`	The name of an `address_match_list` as defined by the **acl** statement.
`address_match_list`	A list of one or more `ip_addr`, `ip_prefix`, `key_id`, or `acl_name` elements, see Section 6.1.1.
`domain_name`	A quoted string which will be used as a DNS name, for example "`my.test.domain`".
`dotted_decimal`	One to four integers valued 0 through 255 separated by dots ('.'), such as **123**, **45.67** or **89.123.45.67**.
`ip4_addr`	An IPv4 address with exactly four elements in `dotted_decimal` notation.
`ip6_addr`	An IPv6 address, such as **2001:db8::1234**. IPv6 scoped addresses that have ambiguity on their scope zones must be disambiguated by an appropriate zone ID with the percent character ('%') as delimiter. It is strongly recommended to use string zone names rather than numeric identifiers, in order to be robust against system configuration changes. However, since there is no standard mapping for such names and identifier values, currently only interface names as link identifiers are supported, assuming one-to-one mapping between interfaces and links. For example, a link-local address **fe80::1** on the link attached to the interface **ne0** can be specified as **fe80::1%ne0**. Note that on most systems link-local addresses always have the ambiguity, and need to be disambiguated.
`ip_addr`	An `ip4_addr` or `ip6_addr`.
`ip_port`	An IP port `number`. `number` is limited to 0 through 65535, with values below 1024 typically restricted to use by processes running as root. In some cases, an asterisk ('*') character can be used as a placeholder to select a random high-numbered port.

ip_prefix	An IP network specified as an ip_addr, followed by a slash ('/') and then the number of bits in the netmask. Trailing zeros in a ip_addr may omitted. For example, **127/8** is the network **127.0.0.0** with netmask **255.0.0.0** and **1.2.3.0/28** is network **1.2.3.0** with netmask **255.255.255.240**.
key_id	A domain_name representing the name of a shared key, to be used for transaction security.
key_list	A list of one or more key_ids, separated by semicolons and ending with a semicolon.
number	A non-negative 32-bit integer (i.e., a number between 0 and 4294967295, inclusive). Its acceptable value might further be limited by the context in which it is used.
path_name	A quoted string which will be used as a pathname, such as zones/master/my.test.domain.
size_spec	A number, the word **unlimited**, or the word **default**. An unlimited size_spec requests unlimited use, or the maximum available amount. A default size_spec uses the limit that was in force when the server was started. A number can optionally be followed by a scaling factor: **K** or **k** for kilobytes, **M** or **m** for megabytes, and **G** or **g** for gigabytes, which scale by 1024, 1024*1024, and 1024*1024*1024 respectively. The value must be representable as a 64-bit unsigned integer (0 to 18446744073709551615, inclusive). Using unlimited is the best way to safely set a really large number.
yes_or_no	Either **yes** or **no**. The words **true** and **false** are also accepted, as are the numbers **1** and **0**.
dialup_option	One of **yes**, **no**, **notify**, **notify-passive**, **refresh** or **passive**. When used in a zone, **notify-passive**, **refresh**, and **passive** are restricted to slave and stub zones.

6.1.1 Address Match Lists

6.1.1.1 Syntax

```
address_match_list = address_match_list_element ;
   address_match_list_element; ...
address_match_list_element =  !  (ip_address /length |
   key key_id | acl_name | { address_match_list } )
```

6.1.1.2 Definition and Usage

Address match lists are primarily used to determine access control for various server operations. They are also used in the **listen-on** and **sortlist** statements. The elements which constitute an address match list can be any of the following:

- an IP address (IPv4 or IPv6)
- an IP prefix (in '/' notation)
- a key ID, as defined by the **key** statement
- the name of an address match list defined with the **acl** statement
- a nested address match list enclosed in braces

Elements can be negated with a leading exclamation mark ('!'), and the match list names "any", "none", "localhost", and "localnets" are predefined. More information on those names can be found in the description of the acl statement.

The addition of the key 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 listen-on ports, or in a sortlist, 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, access is denied. The clauses **allow-notify**, **allow-query**, **allow-transfer**, **allow-update**, **allow-update-forwarding**, and **blackhole** all use address match lists this. Similarly, the listen-on option will cause the server to not accept queries on any of the machine's addresses which do not match the list.

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 **1.2.3/24; ! 1.2.3.13;** 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 **! 1.2.3.13; 1.2.3/24** fixes that problem by having 1.2.3.13 blocked by the negation but all other 1.2.3.* hosts fall through.

6.1.2 Comment Syntax

The BIND 9 comment syntax allows for comments to appear anywhere that white space may appear in a BIND configuration file. To appeal to programmers of all kinds, they can be written in the C, C++, or shell/perl style.

6.1.2.1 Syntax

```
/* This is a BIND comment as in C */

// This is a BIND comment as in C++

# This is a BIND comment as in common UNIX shells and perl
```

6.1.2.2 Definition and Usage

Comments may appear anywhere that white space may appear in a BIND configuration file.

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 # (number sign) and continue to the end of the physical line, as in 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.
```

WARNING

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.

6.2 Configuration File Grammar

A BIND 9 configuration consists of statements and comments. Statements end with a semicolon. Statements and comments are the only elements that can appear without enclosing braces. Many statements contain a block of sub-statements, which are also terminated with a semicolon.

The following statements are supported:

acl	defines a named IP address matching list, for access control and other uses.
controls	declares control channels to be used by the **rndc** utility.
include	includes a file.
key	specifies key information for use in authentication and authorization using TSIG.
logging	specifies what the server logs, and where the log messages are sent.
lwres	configures **named** to also act as a light-weight resolver daemon (**lwresd**).
masters	defines a named masters list for inclusion in stub and slave zone masters clauses.
options	controls global server configuration options and sets defaults for other statements.
server	sets certain configuration options on a per-server basis.
trusted-keys	defines trusted DNSSEC keys.
view	defines a view.
zone	defines a zone.

The **logging** and **options** statements may only occur once per configuration.

6.2.1 acl Statement Grammar

```
acl acl-name {
    address_match_list
};
```

6.2.2 acl Statement Definition and Usage

The **acl** statement assigns a symbolic name to an 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 **acl** before it can be used elsewhere; no forward references are allowed.

The following ACLs are built-in:

any	Matches all hosts.
none	Matches no hosts.
localhost	Matches the IPv4 and IPv6 addresses of all network interfaces on the system.
localnets	Matches any host on an IPv4 or IPv6 network for which the system has an interface. Some systems do not provide a way to determine the prefix lengths of local IPv6 addresses. In such a case, **localnets** only matches the local IPv6 addresses, just like **localhost**.

6.2.3 controls Statement Grammar

```
controls {
   inet ( ip_addr | * )  port ip_port  allow {  address_match_list  }
                keys {  key_list  };
    inet ...;
};
```

6.2.4 controls Statement Definition and Usage

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

An **inet** control channel is a TCP socket listening at the specified **ip_port** on the specified **ip_addr**, which can be an IPv4 or IPv6 address. An **ip_addr** of * (asterisk) is interpreted as the IPv4 wildcard address; connections will be accepted on any of the system's IPv4 addresses. To listen on the IPv6 wildcard address, use an **ip_addr** of `::`. If you will only use **rndc** on the local host, using the loopback address (`127.0.0.1` or `::1`) is recommended for maximum security.

If no port is specified, port 953 is used. The asterisk "*" cannot be used for **ip_port**.

The ability to issue commands over the control channel is restricted by the **allow** and **keys** clauses. Connections to the control channel are permitted based on the **address_match_list**. This is for simple IP address based filtering only; any **key_id** elements of the **address_match_list** are ignored.

The primary authorization mechanism of the command channel is the **key_list**, which contains a list of **key_id**s. Each **key_id** in the **key_list** is authorized to execute commands over the control channel. See

[Remote Name Daemon Control application] in Section 3.3.1.2) for information about configuring keys in **rndc**.

If no **controls** statement is present, **named** will set up a default control channel listening on the loopback address 127.0.0.1 and its IPv6 counterpart ::1. In this case, and also when the **controls** statement is present but does not have a **keys** clause, **named** will attempt to load the command channel key from the file `rndc.key` in `/etc` (or whatever `sysconfdir` was specified as when BIND was built). To create a `rndc.key` file, run **rndc-confgen -a**.

The `rndc.key` feature was created to ease the transition of systems from BIND 8, which did not have digital signatures on its command channel messages and thus did not have a **keys** clause. It makes it possible to use an existing BIND 8 configuration file in BIND 9 unchanged, and still have **rndc** work the same way **ndc** worked in BIND 8, simply by executing the command **rndc-confgen -a** after BIND 9 is installed.

Since the `rndc.key` feature is only intended to allow the backward-compatible usage of BIND 8 configuration files, this feature does not have a high degree of configurability. You cannot easily change the key name or the size of the secret, so you should make a `rndc.conf` with your own key if you wish to change those things. The `rndc.key` file also has its permissions set such that only the owner of the file (the user that **named** is running as) can access it. If you desire greater flexibility in allowing other users to access **rndc** commands, then you need to create a `rndc.conf` file and make it group readable by a group that contains the users who should have access.

The UNIX control channel type of BIND 8 is not supported in BIND 9.0, BIND 9.1, BIND 9.2 and BIND 9.3. If it is present in the controls statement from a BIND 8 configuration file, it is ignored and a warning is logged.

To disable the command channel, use an empty **controls** statement: **controls** { };.

6.2.5 include Statement Grammar

```
include filename;
```

6.2.6 include Statement Definition and Usage

The **include** statement inserts the specified file at the point where the **include** statement is encountered. The **include** statement facilitates the administration of configuration files by permitting the reading or writing of some things but not others. For example, the statement could include private keys that are readable only by the name server.

6.2.7 key Statement Grammar

```
key key_id {
    algorithm string;
    secret string;
};
```

6.2.8 key Statement Definition and Usage

The **key** statement defines a shared secret key for use with TSIG (see Section 4.5) or the command channel (see Section 6.2.4).

The **key** statement can occur at the top level of the configuration file or inside a **view** statement. Keys defined in top-level **key** statements can be used in all views. Keys intended for use in a **controls** statement (see Section 6.2.4) must be defined at the top level.

The *key_id*, also known as the key name, is a domain name uniquely identifying the key. It can be used in a **server** statement to cause requests sent to that server to be signed with this key, or in address match lists to verify that incoming requests have been signed with a key matching this name, algorithm, and secret.

The *algorithm_id* is a string that specifies a security/authentication algorithm. The only algorithm currently supported with TSIG authentication is `hmac-md5`. The *secret_string* is the secret to be used by the algorithm, and is treated as a base-64 encoded string.

6.2.9 logging Statement Grammar

```
logging {
   [ channel channel_name {
     ( file path name
         [ versions ( number | {\verb unlimited} ) ]
         [ size size spec ]
       | syslog syslog_facility
       | stderr
       | 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 ; ... ]
   }; ]
   ...
};
```

6.2.10 logging Statement Definition and Usage

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

Only one **logging** statement is used to define as many channels and categories as are wanted. If there is no **logging** statement, the logging configuration will be:

```
logging {
     category default { default_syslog; default_debug; };
     category unmatched { null; };
};
```

In BIND 9, the logging configuration is only established when the entire configuration file has been parsed. In BIND 8, it was established as soon as the **logging** statement was parsed. When the server is starting up, all logging messages regarding syntax errors in the configuration file go to the default channels, or to standard error if the "`-g`" option was specified.

6.2.10.1 The channel Phrase

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 destination clause that says whether messages selected for the channel go to a file, to a particular syslog facility, to the standard error stream, or are discarded. It can

optionally also limit the message severity level that will be accepted by the channel (the default is **info**), and whether to include a **named**-generated time stamp, the category name and/or severity level (the default is not to include any).

The **null** destination clause causes all messages sent to the channel to be discarded; in that case, other options for the channel are meaningless.

The **file** destination clause directs the channel to a disk file. It 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.

If you use the **versions** log file option, then **named** will retain that many backup versions of the file by renaming them when opening. For example, if you choose to keep three old versions of the file `lamers.log`, then just before it is opened `lamers.log.1` is renamed to `lamers.log.2`, `lamers.log.0` is renamed to `lamers.log.1`, and `lamers.log` is renamed to `lamers.log.0`. You can say **versions unlimited** to not limit the number of versions. If a **size** option is associated with the log file, then renaming is only done when the file being opened exceeds the indicated size. No backup versions are kept by default; any existing log file is simply appended.

The **size** option for files is used to limit log growth. If the file ever exceeds the size, then **named** will stop writing to the file unless it has a **versions** option associated with it. If backup versions are kept, the files are rolled as described above and a new one begun. If there is no **versions** option, no more data will be written to the log until some out-of-band mechanism removes or truncates the log to less than the maximum size. The default behavior is not to limit the size of the file.

Example usage of the **size** and **versions** options:

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

The **syslog** destination clause directs the channel to the system log. Its argument is a syslog facility as described in the **syslog** man page. Known facilities are **kern**, **user**, **mail**, **daemon**, **auth**, **syslog**, **lpr**, **news**, **uucp**, **cron**, **authpriv**, **ftp**, **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6** and **local7**, however not all facilities are supported on all operating systems. How **syslog** will handle messages sent to this facility is described in the **syslog.conf** man page. If you have a system which uses a very old version of **syslog** that only uses two arguments to the **openlog()** function, then this clause is silently ignored.

The **severity** 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 **syslog.conf** priorities will also determine what eventually passes through. For example, defining a channel facility and severity as **daemon** and **debug** but only logging **daemon.warning** via **syslog.conf** will cause messages of severity **info** and **notice** to be dropped. If the situation were reversed, with **named** writing messages of only **warning** or higher, then **syslogd** would print all messages it received from the channel.

The **stderr** destination clause directs the channel to the server's standard error stream. This is intended for use when the server is running as a foreground process, for example when debugging a configuration.

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 **named** server with the `-d` flag followed by a positive integer, or by running **rndc trace**. The global debug level can be set to zero, and debugging mode turned off, by running **rndc notrace**. All debugging messages in the server have a debug level, and higher debug levels give more detailed output. Channels that specify a specific debug severity, for example:

```
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 **dynamic** severity use the server's global debug level to determine what messages to print.

If **print-time** has been turned on, then the date and time will be logged. **print-time** may be specified for a **syslog** channel, but is usually pointless since **syslog** also prints the date and time. If **print-category** is requested, then the category of the message will be logged as well. Finally, if **print-severity** is on, then the severity level of the message will be logged. The **print-** 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 **print-** options are on:

```
28-Feb-2000 15:05:32.863 general:  notice:  running
```

There are four predefined channels that are used for **named**'s default logging as follows. How they are used is described in Section 6.2.10.2.

```
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 {
    stderr;                             // writes to stderr
    severity info;                      // only send priority info
                                        // and higher
};

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

The **default_debug** channel has the special property that it only produces output when the server's debug level is nonzero. It normally writes to a file called `named.run` in the server's working directory.

For security reasons, when the "`-u`" command line option is used, the `named.run` file is created only after **named** has changed to the new UID, and any debug output generated while **named** is starting up and still running as root is discarded. If you need to capture this output, you must run the server with the "`-g`" option and redirect standard error to a file.

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.

6.2.10.2 The category Phrase

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 **default** 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 **null** channel:

```
category xfer-out { null; };
category notify { null; };
```

Following are the available categories and brief descriptions of the types of log information they contain. More categories may be added in future BIND releases.

default	The default category defines the logging options for those categories where no specific configuration has been defined.
general	The catch-all. Many things still aren't classified into categories, and they all end up here.
database	Messages relating to the databases used internally by the name server to store zone and cache data.
security	Approval and denial of requests.
config	Configuration file parsing and processing.
resolver	DNS resolution, such as the recursive lookups performed on behalf of clients by a caching name server.
xfer-in	Zone transfers the server is receiving.
xfer-out	Zone transfers the server is sending.
notify	The NOTIFY protocol.
client	Processing of client requests.
unmatched	Messages that named was unable to determine the class of or for which there was no matching **view**. A one line summary is also logged to the **client** category. This category is best sent to a file or stderr, by default it is sent to the **null** channel.
network	Network operations.
update	Dynamic updates.
update-security	Approval and denial of update requests.

queries	Specify where queries should be logged to.
	At startup, specifying the category **queries** will also enable query logging unless **querylog** option has been specified.
	The query log entry reports the client's IP address and port number, and the query name, class and type. It also reports whether the Recursion Desired flag was set (+ if set, - if not set), EDNS was in use (E) or if the query was signed (S).
	`client 127.0.0.1#62536: query: www.example.com IN AAAA +SE`
	`client ::1#62537: query: www.example.net IN AAAA -SE`

dispatch	Dispatching of incoming packets to the server modules where they are to be processed.
dnssec	DNSSEC and TSIG protocol processing.
lame-servers	Lame servers. These are misconfigurations in remote servers, discovered by BIND 9 when trying to query those servers during resolution.
delegation-only	Delegation only. Logs queries that have have been forced to NXDOMAIN as the result of a delegation-only zone or a **delegation-only** in a hint or stub zone declaration.

6.2.11 lwres Statement Grammar

This is the grammar of the **lwres** statement in the `named.conf` file:

```
lwres {
     listen-on { ip_addr port ip_port ;  ip_addr port ip_port ; ...  };
     view view_name;
     search { domain_name ;  domain_name ; ...  };
     ndots number;
};
```

6.2.12 lwres Statement Definition and Usage

The **lwres** statement configures the name server to also act as a lightweight resolver server. (See Section 5.2.) There may be be multiple **lwres** statements configuring lightweight resolver servers with different properties.

The **listen-on** statement specifies a list of addresses (and ports) that this instance of a lightweight resolver daemon should accept requests on. If no port is specified, port 921 is used. If this statement is omitted, requests will be accepted on 127.0.0.1, port 921.

The **view** statement binds this instance of a lightweight resolver daemon to a view in the DNS namespace, so that the response will be constructed in the same manner as a normal DNS query matching this view. If this statement is omitted, the default view is used, and if there is no default view, an error is triggered.

The **search** statement is equivalent to the **search** statement in `/etc/resolv.conf`. It provides a list of domains which are appended to relative names in queries.

The **ndots** statement is equivalent to the **ndots** statement in `/etc/resolv.conf`. It indicates the minimum number of dots in a relative domain name that should result in an exact match lookup before search path elements are appended.

6.2.13 masters Statement Grammar

```
masters name port ip_port { ( masters_list | ip_addr port ip_port key key ) ; ... } ;
```

6.2.14 masters Statement Definition and Usage

masters lists allow for a common set of masters to be easily used by multiple stub and slave zones.

6.2.15 options Statement Grammar

This is the grammar of the **options** statement in the `named.conf` file:

```
options {
     version version_string;
     hostname hostname_string;
     server-id server_id_string;
     directory path_name;
     key-directory path_name;
     named-xfer path_name;
     tkey-domain domainname;
     tkey-dhkey key_name key_tag;
     dump-file path_name;
     memstatistics-file path_name;
     pid-file path_name;
     statistics-file path_name;
     zone-statistics yes_or_no;
     auth-nxdomain yes_or_no;
     deallocate-on-exit yes_or_no;
     dialup dialup_option;
     fake-iquery yes_or_no;
     fetch-glue yes_or_no;
     flush-zones-on-shutdown yes_or_no;
     has-old-clients yes_or_no;
     host-statistics yes_or_no;
     host-statistics-max number;
     minimal-responses yes_or_no;
     multiple-cnames yes_or_no;
     notify yes_or_no | explicit;
     recursion yes_or_no;
     rfc2308-type1 yes_or_no;
     use-id-pool yes_or_no;
     maintain-ixfr-base yes_or_no;
     dnssec-enable yes_or_no;
     dnssec-lookaside domain trust-anchor domain;
     dnssec-must-be-secure domain yes_or_no;
     forward ( only | first );
     forwarders {  ip_addr port ip_port ; ...  };
     dual-stack-servers port ip_port { ( domain_name port ip_port | ip_addr port ip_port )
     check-names ( master | slave | response )( warn | fail | ignore );
     allow-notify { address_match_list };
     allow-query { address_match_list };
     allow-transfer { address_match_list };
     allow-recursion { address_match_list };
     allow-update-forwarding { address_match_list };
     allow-v6-synthesis { address_match_list };
```

```
blackhole { address_match_list };
avoid-v4-udp-ports { port_list };
avoid-v6-udp-ports { port_list };
listen-on  port ip_port  { address_match_list };
listen-on-v6  port ip_port  { address_match_list };
query-source  address ( ip_addr | * )   port ( ip_port | * ) ;
query-source-v6  address ( ip_addr | * )   port ( ip_port | * ) ;
max-transfer-time-in number;
max-transfer-time-out number;
max-transfer-idle-in number;
max-transfer-idle-out number;
tcp-clients number;
recursive-clients number;
serial-query-rate number;
serial-queries number;
tcp-listen-queue number;
transfer-format ( one-answer | many-answers );
transfers-in  number;
transfers-out number;
transfers-per-ns number;
transfer-source (ip4_addr | *) port ip_port ;
transfer-source-v6 (ip6_addr | *) port ip_port ;
alt-transfer-source (ip4_addr | *) port ip_port ;
alt-transfer-source-v6 (ip6_addr | *) port ip_port ;
use-alt-transfer-source yes_or_no;
notify-source (ip4_addr | *) port ip_port ;
notify-source-v6 (ip6_addr | *) port ip_port ;
also-notify { ip_addr port ip_port ;  ip_addr port ip_port ; ...  };
max-ixfr-log-size number;
max-journal-size size_spec;
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 };
rrset-order { order_spec ;  order_spec ; ...   };
lame-ttl number;
max-ncache-ttl number;
max-cache-ttl number;
sig-validity-interval number ;
min-roots number;
use-ixfr yes_or_no ;
provide-ixfr yes_or_no;
request-ixfr yes_or_no;
treat-cr-as-space yes_or_no ;
min-refresh-time number ;
max-refresh-time number ;
min-retry-time number ;
max-retry-time number ;
port ip_port;
additional-from-auth yes_or_no ;
additional-from-cache yes_or_no ;
random-device path_name ;
max-cache-size size_spec ;
```

```
    match-mapped-addresses yes_or_no;
    preferred-glue ( A | AAAA | NONE );
    edns-udp-size number;
    root-delegation-only  exclude { namelist }  ;
    querylog yes_or_no ;
    disable-algorithms domain { algorithm;  algorithm;  };
};
```

6.2.16 options Statement Definition and Usage

The **options** statement sets up global options to be used by BIND. This statement may appear only once in a configuration file. If there is no **options** statement, an options block with each option set to its default will be used.

directory 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. `named.run`) 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.

key-directory When performing dynamic update of secure zones, the directory where the public and private key files should be found, if different than the current working directory. The directory specified must be an absolute path.

named-xfer *This option is obsolete.* It was used in BIND 8 to specify the pathname to the **named-xfer** program. In BIND 9, no separate **named-xfer** program is needed; its functionality is built into the name server.

tkey-domain The domain appended to the names of all shared keys generated with **TKEY**. When a client requests a **TKEY** exchange, it may or may not specify the desired name for the key. If present, the name of the shared key will be "`client specified part`" + "`tkey-domain`". Otherwise, the name of the shared key will be "`random hex digits`" + "`tkey-domain`". In most cases, the **domainname** should be the server's domain name.

tkey-dhkey The Diffie-Hellman key used by the server to generate shared keys with clients using the Diffie-Hellman mode of **TKEY**. The server must be able to load the public and private keys from files in the working directory. In most cases, the keyname should be the server's host name.

dump-file The pathname of the file the server dumps the database to when instructed to do so with **rndc dumpdb**. If not specified, the default is `named_dump.db`.

memstatistics-file The pathname of the file the server writes memory usage statistics to on exit. If not specified, the default is `named.memstats`.

pid-file The pathname of the file the server writes its process ID in. If not specified, the default is `/var/run/named.pid`. The pid-file is used by programs that want to send signals to the running name server. Specifying **pid-file none** disables the use of a PID file — no file will be written and any existing one will be removed. Note that **none** is a keyword, not a file name, and therefore is not enclosed in double quotes.

statistics-file The pathname of the file the server appends statistics to when instructed to do so using **rndc stats**. If not specified, the default is `named.stats` in the server's current directory. The

format of the file is described in Section 6.2.16.17.

port The UDP/TCP port number the server uses for receiving and sending DNS protocol traffic. The default is 53. This option is mainly intended for server testing; a server using a port other than 53 will not be able to communicate with the global DNS.

random-device The source of entropy to be used by the server. Entropy is primarily needed for DNSSEC operations, such as TKEY transactions and dynamic update of signed zones. This options specifies the device (or file) from which to read entropy. If this is a file, operations requiring entropy will fail when the file has been exhausted. If not specified, the default value is `/dev/random` (or equivalent) when present, and none otherwise. The **random-device** option takes effect during the initial configuration load at server startup time and is ignored on subsequent reloads.

preferred-glue If specified, the listed type (A or AAAA) will be emitted before other glue in the additional section of a query response. The default is not to preference any type (NONE).

root-delegation-only Turn on enforcement of delegation-only in TLDs (top level domains) and root zones with an optional exclude list.

Note some TLDs are not delegation only (e.g. ”DE”, ”LV”, ”US” and ”MUSEUM”).

```
options {
   root-delegation-only exclude { "de"; "lv"; "us"; "museum"; };
};
```

disable-algorithms Disable the specified DNSSEC algorithms at and below the specified name. Multiple **disable-algorithms** statements are allowed. Only the most specific will be applied.

dnssec-lookaside When set, **dnssec-lookaside** provides the validator with an alternate method to validate DNSKEY records at the top of a zone. When a DNSKEY is at or below a domain specified by the deepest **dnssec-lookaside**, and the normal dnssec validation has left the key untrusted, the trust-anchor will be append to the key name and a DLV record will be looked up to see if it can validate the key. If the DLV record validates a DNSKEY (similarly to the way a DS record does) the DNSKEY RRset is deemed to be trusted.

dnssec-must-be-secure Specify heirarchies which must be or may not be secure (signed and validated). If **yes**, then named will only accept answers if they are secure. If **no**, then normal dnssec validation applies allowing for insecure answers to be accepted. The specified domain must be under a **trusted-key** or **dnssec-lookaside** must be active.

6.2.16.1 Boolean Options

auth-nxdomain If **yes**, then the **AA** bit is always set on NXDOMAIN responses, even if the server is not actually authoritative. The default is **no**; this is a change from BIND 8. If you are using very old DNS software, you may need to set it to **yes**.

deallocate-on-exit This option was used in BIND 8 to enable checking for memory leaks on exit. BIND 9 ignores the option and always performs the checks.

dialup If **yes**, 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 **heartbeat-interval** and hopefully during the one call. It also suppresses some of the normal zone maintenance traffic. The default is **no**.

The **dialup** option may also be specified in the **view** and **zone** statements, in which case it overrides the global **dialup** option.

If the zone is a master zone, then the server will send out a NOTIFY request to all the slaves (default). This should trigger the zone serial number check in the slave (providing it supports NOTIFY) allowing the slave to verify the zone while the connection is active. The set of servers to which NOTIFY is sent can be controlled by **notify** and **also-notify**.

If the zone is a slave or stub zone, then the server will suppress the regular "zone up to date" (refresh) queries and only perform them when the **heartbeat-interval** expires in addition to sending NOTIFY requests.

Finer control can be achieved by using **notify** which only sends NOTIFY messages, **notify-passive** which sends NOTIFY messages and suppresses the normal refresh queries, **refresh** which suppresses normal refresh processing and sends refresh queries when the **heartbeat-interval** expires, and **passive** which just disables normal refresh processing.

dialup mode	normal refresh	heart-beat refresh	heart-beat notify
no (default)	yes	no	no
yes	no	yes	yes
notify	yes	no	yes
refresh	no	yes	no
passive	no	no	no
notify-passive	no	no	yes

Note that normal NOTIFY processing is not affected by **dialup**.

fake-iquery In BIND 8, this option enabled simulating the obsolete DNS query type IQUERY. BIND 9 never does IQUERY simulation.

fetch-glue This option is obsolete. In BIND 8, **fetch-glue yes** caused the server to attempt to fetch glue resource records it didn't have when constructing the additional data section of a response. This is now considered a bad idea and BIND 9 never does it.

flush-zones-on-shutdown When the nameserver exits due receiving SIGTERM, flush or do not flush any pending zone writes. The default is **flush-zones-on-shutdown no**.

has-old-clients This option was incorrectly implemented in BIND 8, and is ignored by BIND 9. To achieve the intended effect of **has-old-clients yes**, specify the two separate options **auth-nxdomain yes** and **rfc2308-type1 no** instead.

host-statistics In BIND 8, this enables keeping of statistics for every host that the name server interacts with. Not implemented in BIND 9.

maintain-ixfr-base *This option is obsolete*. It was used in BIND 8 to determine whether a transaction log was kept for Incremental Zone Transfer. BIND 9 maintains a transaction log whenever possible. If you need to disable outgoing incremental zone transfers, use **provide-ixfr no**.

minimal-responses If **yes**, then when generating responses the server will only add records to the authority and additional data sections when they are required (e.g. delegations, negative responses). This may improve the performance of the server. The default is **no**.

multiple-cnames This option was used in BIND 8 to allow a domain name to have multiple CNAME records in violation of the DNS standards. BIND 9.2 always strictly enforces the CNAME rules both in master files and dynamic updates.

notify If **yes** (the default), DNS NOTIFY messages are sent when a zone the server is authoritative for changes, see Section 4.1. The messages are sent to the servers listed in the zone's NS records (except the master server identified in the SOA MNAME field), and to any servers listed in the **also-notify** option.

If **explicit**, notifies are sent only to servers explicitly listed using **also-notify**. If **no**, no notifies are sent.

The **notify** option may also be specified in the **zone** statement, in which case it overrides the **options notify** statement. It would only be necessary to turn off this option if it caused slaves to crash.

recursion If **yes**, and a DNS query requests recursion, then the server will attempt to do all the work required to answer the query. If recursion is off and the server does not already know the answer, it will return a referral response. The default is **yes**. Note that setting **recursion no** does not prevent clients from getting data from the server's cache; it only prevents new data from being cached as an effect of client queries. Caching may still occur as an effect the server's internal operation, such as NOTIFY address lookups. See also **fetch-glue** above.

rfc2308-type1 Setting this to **yes** will cause the server to send NS records along with the SOA record for negative answers. The default is **no**.

NOTE

Not yet implemented in BIND 9.

use-id-pool *This option is obsolete*. BIND 9 always allocates query IDs from a pool.

zone-statistics If **yes**, the server will collect statistical data on all zones (unless specifically turned off on a per-zone basis by specifying **zone-statistics no** in the **zone** statement). These statistics may be accessed using **rndc stats**, which will dump them to the file listed in the **statistics-file**. See also Section 6.2.16.17.

use-ixfr *This option is obsolete*. If you need to disable IXFR to a particular server or servers see the information on the **provide-ixfr** option in Section 6.2.18. See also Section 4.3.

provide-ixfr See the description of **provide-ixfr** in Section 6.2.18.

request-ixfr See the description of **request-ixfr** in Section 6.2.18.

treat-cr-as-space This option was used in BIND 8 to make the server treat carriage return ("**r**") characters the same way as a space or tab character, to facilitate loading of zone files on a UNIX system that were generated on an NT or DOS machine. In BIND 9, both UNIX "**n**" and NT/DOS "**r****n**" newlines are always accepted, and the option is ignored.

additional-from-auth, additional-from-cache These options control the behavior of an authoritative server when answering queries which have additional data, or when following CNAME and DNAME chains.

When both of these options are set to **yes** (the default) and a query is being answered from authoritative data (a zone configured into the server), the additional data section of the reply will be filled in using data from other authoritative zones and from the cache. In some situations this is undesirable, such as when there is concern over the correctness of the cache, or in servers where slave zones may be added and modified by untrusted third parties. Also, avoiding the search for this additional data will speed up server operations at the possible expense of additional queries to resolve what would otherwise be provided in the additional section.

For example, if a query asks for an MX record for host `foo.example.com`, and the record found is "`MX 10 mail.example.net`", normally the address records (A and AAAA) for `mail.example.net` will be provided as well, if known, even though they are not in the example.com zone. Setting these options to **no** disables this behavior and makes the server only search for additional data in the zone it answers from.

These options are intended for use in authoritative-only servers, or in authoritative-only views. Attempts to set them to **no** without also specifying **recursion no** will cause the server to ignore the options and log a warning message.

Specifying **additional-from-cache no** actually disables the use of the cache not only for additional data lookups but also when looking up the answer. This is usually the desired behavior in an authoritative-only server where the correctness of the cached data is an issue.

When a name server is non-recursively queried for a name that is not below the apex of any served zone, it normally answers with an "upwards referral" to the root servers or the servers of some other known parent of the query name. Since the data in an upwards referral comes from the cache, the server will not be able to provide upwards referrals when **additional-from-cache no** has been specified. Instead, it will respond to such queries with REFUSED. This should not cause any problems since upwards referrals are not required for the resolution process.

match-mapped-addresses If **yes**, then an IPv4-mapped IPv6 address will match any address match list entries that match the corresponding IPv4 address. Enabling this option is sometimes useful on IPv6-enabled Linux systems, to work around a kernel quirk that causes IPv4 TCP connections such as zone transfers to be accepted on an IPv6 socket using mapped addresses, causing address match lists designed for IPv4 to fail to match. The use of this option for any other purpose is discouraged.

ixfr-from-differences When **yes** and the server loads a new version of a master zone from its zone file or receives a new version of a slave file by a non-incremental zone transfer, it will compare the new version to the previous one and calculate a set of differences. The differences are then logged in the zone's journal file such that the changes can be transmitted to downstream slaves as an incremental zone transfer.

By allowing incremental zone transfers to be used for non-dynamic zones, this option saves bandwidth at the expense of increased CPU and memory consumption at the master. In particular, if the new version of a zone is completely different from the previous one, the set of differences will be of a size comparable to the combined size of the old and new zone version, and the server will need to temporarily allocate memory to hold this complete difference set.

multi-master This should be set when you have multiple masters for a zone and the addresses refer to different machines. If **yes**, named will not log when the serial number on the master is less than what named currently has. The default is **no**.

dnssec-enable Enable DNSSEC support in named. Unless set to **yes**, named behaves as if it does not support DNSSEC. The default is **no**.

querylog Specify whether query logging should be started when named starts. If **querylog** is not specified, then the query logging is determined by the presence of the logging category **queries**.

check-names This option is used to restrict the character set and syntax of certain domain names in master files and/or DNS responses received from the network. The default varies according to usage area. For **master** zones the default is **fail**. For **slave** zones the default is **warn**. For answers received from the network (**response**) the default is **ignore**.

The rules for legal hostnames and mail domains are derived from RFC 952 and RFC 821 as modified by RFC 1123.

check-names applies to the owner names of A, AAA and MX records. It also applies to the domain names in the RDATA of NS, SOA and MX records. It also applies to the RDATA of PTR records where the owner name indicated that it is a reverse lookup of a hostname (the owner name ends in IN-ADDR.ARPA, IP6.ARPA, IP6.INT).

6.2.16.2 Forwarding

The forwarding facility can be used to create a large site-wide cache on a few servers, reducing traffic over links to external name servers. 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.

forward This option is only meaningful if the forwarders list is not empty. A value of `first`, 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 `only` is specified, the server will only query the forwarders.

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-domain basis, allowing for the global forwarding options to be overridden in a variety of ways. You can set particular domains to use different forwarders, or have a different **forward only/first** behavior, or not forward at all, see Section 6.2.23.

6.2.16.3 Dual-stack Servers

Dual-stack servers are used as servers of last resort to work around problems in reachability due the lack of support for either IPv4 or IPv6 on the host machine.

dual-stack-servers Specifies host names or addresses of machines with access to both IPv4 and IPv6 transports. If a hostname is used, the server must be able to resolve the name using only the transport it has. If the machine is dual stacked, then the **dual-stack-servers** have no effect unless access to a transport has been disabled on the command line (e.g. **named -4**).

6.2.16.4 Access Control

Access to the server can be restricted based on the IP address of the requesting system. See Section 6.1.1 for details on how to specify IP address lists.

allow-notify Specifies which hosts are allowed to notify this server, a slave, of zone changes in addition to the zone masters. **allow-notify** may also be specified in the **zone** statement, in which case it overrides the **options allow-notify** statement. It is only meaningful for a slave zone. If not specified, the default is to process notify messages only from a zone's master.

allow-query Specifies which hosts are allowed to ask ordinary DNS questions. **allow-query** may also be specified in the **zone** statement, in which case it overrides the **options allow-query** statement. If not specified, the default is to allow queries from all hosts.

allow-recursion Specifies which hosts are allowed to make recursive queries through this server. If not specified, the default is to allow recursive queries from all hosts. Note that disallowing recursive queries for a host does not prevent the host from retrieving data that is already in the server's cache.

allow-update-forwarding Specifies which hosts are allowed to submit Dynamic DNS updates to slave zones to be forwarded to the master. The default is **`{ none; }`**, which means that no update forwarding will be performed. To enable update forwarding, specify **`allow-update-forwarding { any; };`**. Specifying values other than **`{ none; }`** or **`{ any; }`** is usually counterproductive, since the responsibility for update access control should rest with the master server, not the slaves.

Note that enabling the update forwarding feature on a slave server may expose master servers relying on insecure IP address based access control to attacks; see Section 7.3 for more details.

allow-v6-synthesis This option was introduced for the smooth transition from AAAA to A6 and from "nibble labels" to binary labels. However, since both A6 and binary labels were then deprecated, this option was also deprecated. It is now ignored with some warning messages.

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

blackhole 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 default is **`none`**.

6.2.16.5 Interfaces

The interfaces and ports that the server will answer queries from may be specified using the **listen-on** option. **listen-on** 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 **listen-on** 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 name server 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 **listen-on** is specified, the server will listen on port 53 on all interfaces.

The **listen-on-v6** option is used to specify the interfaces and the ports on which the server will listen for incoming queries sent using IPv6.

When

```
{ any; }
```

is specified as the `address_match_list` for the **listen-on-v6** option, the server does not bind a separate socket to each IPv6 interface address as it does for IPv4 if the operating system has enough API support for IPv6 (specifically if it conforms to RFC 3493 and RFC 3542). Instead, it listens on the IPv6 wildcard

address. If the system only has incomplete API support for IPv6, however, the behavior is the same as that for IPv4.

A list of particular IPv6 addresses can also be specified, in which case the server listens on a separate socket for each specified address, regardless of whether the desired API is supported by the system.

Multiple **listen-on-v6** options can be used. For example,

```
listen-on-v6 { any; };
listen-on-v6 port 1234 { !2001:db8::/32; any; };
```

will enable the name server on port 53 for any IPv6 addresses (with a single wildcard socket), and on port 1234 of IPv6 addresses that is not in the prefix 2001:db8::/32 (with separate sockets for each matched address.)

To make the server not listen on any IPv6 address, use

```
listen-on-v6 { none; };
```

If no **listen-on-v6** option is specified, the server will not listen on any IPv6 address.

6.2.16.6 Query Address

If the server doesn't know the answer to a question, it will query other name servers. **query-source** specifies the address and port used for such queries. For queries sent over IPv6, there is a separate **query-source-v6** option. If **address** is * (asterisk) or is omitted, a wildcard IP address (**INADDR_ANY**) will be used. If **port** is * or is omitted, a random unprivileged port will be used. The **avoid-v4-udp-ports** and **avoid-v6-udp-ports** options can be used to prevent named from selecting certain ports. The defaults are:

```
query-source address * port *;
query-source-v6 address * port *;
```

NOTE

The address specified in the **query-source** option is used for both UDP and TCP queries, but the port applies only to UDP queries. TCP queries always use a random unprivileged port.

NOTE

See also **transfer-source** and **notify-source**.

NOTE

Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

6.2.16.7 Zone Transfers

BIND has mechanisms in place to facilitate zone transfers and set limits on the amount of load that transfers place on the system. The following options apply to zone transfers.

also-notify Defines a global list of IP addresses of name servers that are also sent NOTIFY messages whenever a fresh copy of the zone is loaded, in addition to the servers listed in the zone's NS records. This helps to ensure that copies of the zones will quickly converge on stealth servers. If an **also-notify** list is given in a **zone** statement, it will override the **options also-notify** statement. When a **zone notify** statement is set to **no**, the IP addresses in the global **also-notify** list will not be sent NOTIFY messages for that zone. The default is the empty list (no global notification list).

max-transfer-time-in Inbound zone transfers running longer than this many minutes will be terminated. The default is 120 minutes (2 hours). The maximum value is 28 days (40320 minutes).

max-transfer-idle-in Inbound zone transfers making no progress in this many minutes will be terminated. The default is 60 minutes (1 hour). The maximum value is 28 days (40320 minutes).

max-transfer-time-out Outbound zone transfers running longer than this many minutes will be terminated. The default is 120 minutes (2 hours). The maximum value is 28 days (40320 minutes).

max-transfer-idle-out Outbound zone transfers making no progress in this many minutes will be terminated. The default is 60 minutes (1 hour). The maximum value is 28 days (40320 minutes).

serial-query-rate Slave servers will periodically query master servers to find out if zone serial numbers have changed. Each such query uses a minute amount of the slave server's network bandwidth. To limit the amount of bandwidth used, BIND 9 limits the rate at which queries are sent. The value of the **serial-query-rate** option, an integer, is the maximum number of queries sent per second. The default is 20.

serial-queries In BIND 8, the **serial-queries** option set the maximum number of concurrent serial number queries allowed to be outstanding at any given time. BIND 9 does not limit the number of outstanding serial queries and ignores the **serial-queries** option. Instead, it limits the rate at which the queries are sent as defined using the **serial-query-rate** option.

transfer-format Zone transfers can be sent using two different formats, **one-answer** and **many-answers**. The **transfer-format** option is used on the master server to determine which format it sends. **one-answer** uses one DNS message per resource record transferred. **many-answers** packs as many resource records as possible into a message. **many-answers** is more efficient, but is only supported by relatively new slave servers, such as BIND 9, BIND 8.x and patched versions of BIND 4.9.5. The **many-answers** format is also supported by recent Microsoft Windows nameservers. The default is **many-answers**. **transfer-format** may be overridden on a per-server basis by using the **server** statement.

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

transfers-out The maximum number of outbound zone transfers that can be running concurrently. Zone transfer requests in excess of the limit will be refused. The default value is `10`.

transfers-per-ns The maximum number of inbound zone transfers that can be concurrently transferring from a given remote name server. The default value is `2`. Increasing **transfers-per-ns** may speed

up the convergence of slave zones, but it also may increase the load on the remote name server. **transfers-per-ns** may be overridden on a per-server basis by using the **transfers** phrase of the **server** statement.

transfer-source **transfer-source** determines which local address will be bound to IPv4 TCP connections used to fetch zones transferred inbound by the server. It also determines the source IPv4 address, and optionally the UDP port, used for the refresh queries and forwarded dynamic updates. 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 **allow-transfer** option for the zone being transferred, if one is specified. This statement sets the **transfer-source** for all zones, but can be overridden on a per-view or per-zone basis by including a **transfer-source** statement within the **view** or **zone** block in the configuration file.

transfer-source-v6 The same as **transfer-source**, except zone transfers are performed using IPv6.

alt-transfer-source An alternate transfer source if the one listed in **transfer-source** fails and **use-alt-transfer-source** is set.

NOTE

If you do not wish the alternate transfer source to be used, you should set **use-alt-transfer-source** appropriately and you should not depend upon getting a answer back to the first refresh query.

alt-transfer-source-v6 An alternate transfer source if the one listed in **transfer-source-v6** fails and **use-alt-transfer-source** is set.

use-alt-transfer-source Use the alternate transfer sources or not. If views are specified this defaults to **no** otherwise it defaults to **yes** (for BIND 8 compatibility).

notify-source **notify-source** determines which local source address, and optionally UDP port, will be used to send NOTIFY messages. This address must appear in the slave server's **masters** zone clause or in an **allow-notify** clause. This statement sets the **notify-source** for all zones, but can be overridden on a per-zone or per-view basis by including a **notify-source** statement within the **zone** or **view** block in the configuration file.

NOTE

Solaris 2.5.1 and earlier does not support setting the source address for TCP sockets.

notify-source-v6 Like **notify-source**, but applies to notify messages sent to IPv6 addresses.

6.2.16.8 Bad UDP Port Lists

avoid-v4-udp-ports and **avoid-v6-udp-ports** specify a list of IPv4 and IPv6 UDP ports that will not be used as system assigned source ports for UDP sockets. These lists prevent named from choosing as its

random source port a port that is blocked by your firewall. If a query went out with such a source port, the answer would not get by the firewall and the name server would have to query again.

6.2.16.9 Operating System Resource Limits

The server's usage of many system resources can be limited. Scaled values are allowed when specifying resource limits. For example, **1G** can be used instead of **1073741824** to specify a limit of one gigabyte. **unlimited** requests unlimited use, or the maximum available amount. **default** uses the limit that was in force when the server was started. See the description of **size_spec** in Section 6.1.

The following options set operating system resource limits for the name server process. Some operating systems don't support some or any of the limits. On such systems, a warning will be issued if the unsupported limit is used.

coresize The maximum size of a core dump. The default is `default`.

datasize The maximum amount of data memory the server may use. The default is `default`. This is a hard limit on server memory usage. If the server attempts to allocate memory in excess of this limit, the allocation will fail, which may in turn leave the server unable to perform DNS service. Therefore, this option is rarely useful as a way of limiting the amount of memory used by the server, but it can be used to raise an operating system data size limit that is too small by default. If you wish to limit the amount of memory used by the server, use the **max-cache-size** and **recursive-clients** options instead.

files The maximum number of files the server may have open concurrently. The default is `unlimited`.

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

6.2.16.10 Server Resource Limits

The following options set limits on the server's resource consumption that are enforced internally by the server rather than the operating system.

max-ixfr-log-size This option is obsolete; it is accepted and ignored for BIND 8 compatibility. The option **max-journal-size** performs a similar function in BIND 8.

max-journal-size Sets a maximum size for each journal file (see Section 4.2.1). When the journal file approaches the specified size, some of the oldest transactions in the journal will be automatically removed. The default is `unlimited`.

host-statistics-max In BIND 8, specifies the maximum number of host statistics entries to be kept. Not implemented in BIND 9.

recursive-clients The maximum number of simultaneous recursive lookups the server will perform on behalf of clients. The default is `1000`. Because each recursing client uses a fair bit of memory, on the order of 20 kilobytes, the value of the **recursive-clients** option may have to be decreased on hosts with limited memory.

tcp-clients The maximum number of simultaneous client TCP connections that the server will accept. The default is `100`.

max-cache-size The maximum amount of memory to use for the server's cache, in bytes. When the amount of data in the cache reaches this limit, the server will cause records to expire prematurely so that the limit is not exceeded. In a server with multiple views, the limit applies separately to the cache of each view. The default is `unlimited`, meaning that records are purged from the cache only when their TTLs expire.

tcp-listen-queue The listen queue depth. The default and minimum is 3. If the kernel supports the accept filter "dataready" this also controls how many TCP connections that will be queued in kernel space waiting for some data before being passed to accept. Values less than 3 will be silently raised.

6.2.16.11 Periodic Task Intervals

cleaning-interval The server will remove expired resource records from the cache every **cleaning-interval** minutes. The default is 60 minutes. The maximum value is 28 days (40320 minutes). If set to 0, no periodic cleaning will occur.

heartbeat-interval The server will perform zone maintenance tasks for all zones marked as **dialup** whenever this interval expires. The default is 60 minutes. Reasonable values are up to 1 day (1440 minutes). The maximum value is 28 days (40320 minutes). If set to 0, no zone maintenance for these zones will occur.

interface-interval The server will scan the network interface list every **interface-interval** minutes. The default is 60 minutes. The maximum value is 28 days (40320 minutes). If set to 0, interface scanning will only occur when the configuration file is loaded. After the scan, the server will begin listening for queries on any newly discovered interfaces (provided they are allowed by the **listen-on** configuration), and will stop listening on interfaces that have gone away.

statistics-interval Name server statistics will be logged every **statistics-interval** minutes. The default is 60. The maximum value is 28 days (40320 minutes). If set to 0, no statistics will be logged.

> NOTE
>
> 
>
> Not yet implemented in BIND9.

6.2.16.12 Topology

All other things being equal, when the server chooses a name server to query from a list of name servers, it prefers the one that is topologically closest to itself. The **topology** 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

```
topology { localhost; localnets; };
```

NOTE

The **topology** option is not implemented in BIND 9.

6.2.16.13 The sortlist Statement

The response to a DNS query may consist of multiple resource records (RRs) forming a resource records set (RRset). The name server will normally return the RRs within the RRset in an indeterminate order (but see the **rrset-order** statement in Section 6.2.16.14). The client resolver code should rearrange the RRs as appropriate, that is, using any addresses on the local net in preference to other addresses. However, not all resolvers can do this or are 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 name servers, not all the clients.

The **sortlist** statement (see below) takes an **address_match_list** and interprets it even more specifically than the **topology** statement does (Section 6.2.16.12). 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 a 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, then the second element is treated the same as 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 behavior 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; };
};
```

6.2.16.14 RRset Ordering

When multiple records are returned in an answer it may be useful to configure the order of the records placed into the response. The **rrset-order** statement permits configuration of the ordering of the records in a multiple record response. See also the **sortlist** statement, Section 6.2.16.13.

An **order_spec** is defined as follows:

```
 class class_name  type type_name  name "domain_name"
      order ordering
```

If no class is specified, the default is **ANY**. If no type is specified, the default is **ANY**. If no name is specified, the default is "*" (asterisk).

The legal values for **ordering** are:

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

For example:

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

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

If multiple **rrset-order** statements appear, they are not combined — the last one applies.

NOTE

The **rrset-order** statement is not yet fully implemented in BIND 9. BIND 9 currently does not support "fixed" ordering.

6.2.16.15 Tuning

lame-ttl Sets the number of seconds to cache a lame server indication. 0 disables caching. (This is **NOT** recommended.) The default is `600` (10 minutes) and the maximum value is `1800` (30 minutes).

max-ncache-ttl To reduce network traffic and increase performance, the server stores negative answers. **max-ncache-ttl** is used to set a maximum retention time for these answers in the server in seconds. The default **max-ncache-ttl** is `10800` seconds (3 hours). **max-ncache-ttl** cannot exceed 7 days and will be silently truncated to 7 days if set to a greater value.

max-cache-ttl Sets the maximum time for which the server will cache ordinary (positive) answers. The default is one week (7 days).

min-roots The minimum number of root servers that is required for a request for the root servers to be accepted. The default is **2**.

> NOTE
>
> Not implemented in BIND 9.

sig-validity-interval Specifies the number of days into the future when DNSSEC signatures automatically generated as a result of dynamic updates (Section 4.2) will expire. The default is `30` days. The maximum value is 10 years (3660 days). The signature inception time is unconditionally set to one hour before the current time to allow for a limited amount of clock skew.

min-refresh-time, max-refresh-time, min-retry-time, max-retry-time These options control the server's behavior on refreshing a zone (querying for SOA changes) or retrying failed transfers. Usually the SOA values for the zone are used, but these values are set by the master, giving slave server administrators little control over their contents.

These options allow the administrator to set a minimum and maximum refresh and retry time either per-zone, per-view, or globally. These options are valid for slave and stub zones, and clamp the SOA refresh and retry times to the specified values.

edns-udp-size **edns-udp-size** sets the advertised EDNS UDP buffer size in bytes. Valid values are 512 to 4096 bytes (values outside this range will be silently adjusted). The default value is 4096. The usual reason for setting edns-udp-size to a non-default value it to get UDP answers to pass through broken firewalls that block fragmented packets and/or block UDP packets that are greater than 512 bytes.

6.2.16.16 Built-in server information zones

The server provides some helpful diagnostic information through a number of built-in zones under the pseudo-top-level-domain `bind` in the **CHAOS** class. These zones are part of a built-in view (see Section 6.2.21) of class **CHAOS** which is separate from the default view of class **IN**; therefore, any global server options such as **allow-query** do not apply the these zones. If you feel the need to disable these zones, use the options below, or hide the built-in **CHAOS** view by defining an explicit view of class **CHAOS** that matches all clients.

version The version the server should report via a query of the name `version.bind` with type **TXT**, class **CHAOS**. The default is the real version number of this server. Specifying **version none** disables processing of the queries.

hostname The hostname the server should report via a query of the name `hostname.bind` with type **TXT**, class **CHAOS**. This defaults to the hostname of the machine hosting the name server as found by the gethostname() function. The primary purpose of such queries is to identify which of a group of anycast servers is actually answering your queries. Specifying **hostname none;** disables processing of the queries.

server-id The ID of the server should report via a query of the name `ID.SERVER` with type **TXT**, class **CHAOS**. The primary purpose of such queries is to identify which of a group of anycast servers is actually answering your queries. Specifying **server-id none;** disables processing of the queries. Specifying **server-id hostname;** will cause named to use the hostname as found by the gethostname() function. The default **server-id** is **none**.

6.2.16.17 The Statistics File

The statistics file generated by BIND 9 is similar, but not identical, to that generated by BIND 8.

The statistics dump begins with a line, like:

+++ Statistics Dump +++ (973798949)

The numberr in parentheses is a standard Unix-style timestamp, measured as seconds since January 1, 1970. Following that line are a series of lines containing a counter type, the value of the counter, optionally a zone name, and optionally a view name. The lines without view and zone listed are global statistics for the entire server. Lines with a zone and view name for the given view and zone (the view name is omitted for the default view).

The statistics dump ends with the line where the number is identical to the number in the beginning line; for example:

— Statistics Dump — (973798949)

The following statistics counters are maintained:

success	The number of successful queries made to the server or zone. A successful query is defined as query which returns a NOERROR response with at least one answer RR.
referral	The number of queries which resulted in referral responses.
nxrrset	The number of queries which resulted in NOERROR responses with no data.
nxdomain	The number of queries which resulted in NXDOMAIN responses.
failure	The number of queries which resulted in a failure response other than those above.
recursion	The number of queries which caused the server to perform recursion in order to find the final answer.

Each query received by the server will cause exactly one of **success**, **referral**, **nxrrset**, **nxdomain**, or **failure** to be incremented, and may additionally cause the **recursion** counter to be incremented.

6.2.17 server Statement Grammar

```
server ip_addr {
     bogus yes_or_no ;
     provide-ixfr yes_or_no ;
     request-ixfr yes_or_no ;
     edns yes_or_no ;
     transfers number ;
     transfer-format ( one-answer | many-answers ) ; ]
     keys { string ;  string ; ... } ;
     transfer-source (ip4_addr | *) port ip_port ;
     transfer-source-v6 (ip6_addr | *) port ip_port ;
};
```

6.2.18 server Statement Definition and Usage

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

The **server** statement can occur at the top level of the configuration file or inside a **view** statement. If a **view** statement contains one or more **server** statements, only those apply to the view and any top-level ones are ignored. If a view contains no **server** statements, any top-level **server** statements are used as defaults.

If you discover that a remote server is giving out bad data, marking it as bogus will prevent further queries to it. The default value of **bogus** is **no**.

The **provide-ixfr** clause determines whether the local server, acting as master, will respond with an incremental zone transfer when the given remote server, a slave, requests it. If set to **yes**, incremental transfer will be provided whenever possible. If set to **no**, all transfers to the remote server will be non-incremental. If not set, the value of the **provide-ixfr** option in the view or global options block is used as a default.

The **request-ixfr** clause determines whether the local server, acting as a slave, will request incremental zone transfers from the given remote server, a master. If not set, the value of the **request-ixfr** option in the view or global options block is used as a default.

IXFR requests to servers that do not support IXFR will automatically fall back to AXFR. Therefore, there is no need to manually list which servers support IXFR and which ones do not; the global default of **yes** should always work. The purpose of the **provide-ixfr** and **request-ixfr** clauses is to make it possible to disable the use of IXFR even when both master and slave claim to support it, for example if one of the servers is buggy and crashes or corrupts data when IXFR is used.

The **edns** clause determines whether the local server will attempt to use EDNS when communicating with the remote server. The default is **yes**.

The server supports two zone transfer methods. The first, **one-answer**, uses one DNS message per resource record transferred. **many-answers** packs as many resource records as possible into a message. **many-answers** is more efficient, but is only known to be understood by BIND 9, BIND 8.x, and patched versions of BIND 4.9.5. You can specify which method to use for a server with the **transfer-format** option. If **transfer-format** is not specified, the **transfer-format** specified by the **options** statement will be used.

transfers is used to limit the number of concurrent inbound zone transfers from the specified server. If no **transfers** clause is specified, the limit is set according to the **transfers-per-ns** option.

The **keys** clause identifies a **key_id** defined by the **key** statement, to be used for transaction security (TSIG, Section 4.5) when talking to the remote server. When a request is sent to the remote server, a request signature will be generated using the key specified here and appended to the message. A request originating from the remote server is not required to be signed by this key.

Although the grammar of the **keys** clause allows for multiple keys, only a single key per server is currently supported.

The **transfer-source** and **transfer-source-v6** clauses specify the IPv4 and IPv6 source address to be used for zone transfer with the remote server, respectively. For an IPv4 remote server, only **transfer-source** can be specified. Similarly, for an IPv6 remote server, only **transfer-source-v6** can be specified. For more details, see the description of **transfer-source** and **transfer-source-v6** in Section 6.2.16.7.

6.2.19 trusted-keys Statement Grammar

```
trusted-keys {
    string number number number string ;
     string number number number string ; ...
};
```

6.2.20 trusted-keys Statement Definition and Usage

The **trusted-keys** statement defines DNSSEC security roots. DNSSEC is described in Section 4.8. A security root is defined when the public key for a non-authoritative zone is known, but cannot be securely obtained through DNS, either because it is the DNS root zone or because its parent zone is unsigned. Once a key has been configured as a trusted key, it is treated as if it had been validated and proven secure. The resolver attempts DNSSEC validation on all DNS data in subdomains of a security root.

All keys (and corresponding zones) listed in **trusted-keys** are deemed to exist regardless of what parent zones say. Similarly for all keys listed in **trusted-keys** only those keys are used to validate the DNSKEY RRset. The parent's DS RRset will not be used.

The **trusted-keys** statement can contain multiple key entries, each consisting of the key's domain name, flags, protocol, algorithm, and the Base-64 representation of the key data.

6.2.21 view Statement Grammar

```
view view_name
      class {
      match-clients { address_match_list } ;
      match-destinations { address_match_list } ;
      match-recursive-only yes_or_no ;
       view_option; ...
       zone_statement; ...
};
```

6.2.22 view Statement Definition and Usage

The **view** statement is a powerful new feature of BIND 9 that lets a name server answer a DNS query differently depending on who is asking. It is particularly useful for implementing split DNS setups without having to run multiple servers.

Each **view** statement defines a view of the DNS namespace that will be seen by a subset of clients. A client matches a view if its source IP address matches the `address_match_list` of the view's **match-clients** clause and its destination IP address matches the `address_match_list` of the view's **match-destinations** clause. If not specified, both **match-clients** and **match-destinations** default to matching all addresses. In addition to checking IP addresses **match-clients** and **match-destinations** can also take **keys** which provide an mechanism for the client to select the view. A view can also be specified as **match-recursive-only**, which means that only recursive requests from matching clients will match that view. The order of the **view** statements is significant — a client request will be resolved in the context of the first **view** that it matches.

Zones defined within a **view** statement will be only be accessible to clients that match the **view**. By defining a zone of the same name in multiple views, different zone data can be given to different clients, for example, "internal" and "external" clients in a split DNS setup.

Many of the options given in the **options** statement can also be used within a **view** statement, and then apply only when resolving queries with that view. When no view-specific value is given, the value in the **options** statement is used as a default. Also, zone options can have default values specified in the **view** statement; these view-specific defaults take precedence over those in the **options** statement.

Views are class specific. If no class is given, class IN is assumed. Note that all non-IN views must contain a hint zone, since only the IN class has compiled-in default hints.

If there are no **view** statements in the config file, a default view that matches any client is automatically created in class IN. Any **zone** statements specified on the top level of the configuration file are considered to be part of this default view, and the **options** statement will apply to the default view. If any explicit **view** statements are present, all **zone** statements must occur inside **view** statements.

Here is an example of a typical split DNS setup implemented using **view** statements:

```
view "internal" {
      // This should match our internal networks.
      match-clients { 10.0.0.0/8; };

      // Provide recursive service to internal clients only.
      recursion yes;

      // Provide a complete view of the example.com zone
      // including addresses of internal hosts.
      zone "example.com" {
            type master;
            file "example-internal.db";
      };
};

view "external" {
      // Match all clients not matched by the previous view.
      match-clients { any; };

      // Refuse recursive service to external clients.
      recursion no;

      // Provide a restricted view of the example.com zone
      // containing only publicly accessible hosts.
      zone "example.com" {
           type master;
           file "example-external.db";
      };
};
```

6.2.23 zone Statement Grammar

```
zone zone_name class {
    type master;
     allow-query { address_match_list } ;
     allow-transfer { address_match_list } ;
     allow-update { address_match_list } ;
     update-policy { update_policy_rule ... } ;
     also-notify { ip_addr port ip_port ;  ip_addr port ip_port ; ...  };
     check-names (warn|fail|ignore) ;
```

```
     dialup dialup_option ;
     file string ;
     forward (only|first) ;
     forwarders {  ip_addr port ip_port ; ...  };
     ixfr-base string ;
     ixfr-tmp-file string ;
     maintain-ixfr-base yes_or_no ;
     max-ixfr-log-size number ;
     max-transfer-idle-out number ;
     max-transfer-time-out number ;
     notify yes_or_no | explicit ;
     pubkey number number number string ;
     notify-source (ip4_addr | *) port ip_port ;
     notify-source-v6 (ip6_addr | *) port ip_port ;
     zone-statistics yes_or_no ;
     sig-validity-interval number ;
     database string ;
     min-refresh-time number ;
     max-refresh-time number ;
     min-retry-time number ;
     max-retry-time number ;
     key-directory path_name;
};

zone zone_name class {
    type slave;
     allow-notify { address_match_list } ;
     allow-query { address_match_list } ;
     allow-transfer { address_match_list } ;
     allow-update-forwarding { address_match_list } ;
     also-notify { ip_addr port ip_port ;  ip_addr port ip_port ; ...  };
     check-names (warn|fail|ignore) ;
     dialup dialup_option ;
     file string ;
     forward (only|first) ;
     forwarders {  ip_addr port ip_port ; ...  };
     ixfr-base string ;
     ixfr-tmp-file string ;
     maintain-ixfr-base yes_or_no ;
     masters port ip_port { ( masters_list | ip_addr port ip_port key key ) ; ... } ;
     max-ixfr-log-size number ;
     max-transfer-idle-in number ;
     max-transfer-idle-out number ;
     max-transfer-time-in number ;
     max-transfer-time-out number ;
     notify yes_or_no | explicit ;
     pubkey number number number string ;
     transfer-source (ip4_addr | *) port ip_port ;
     transfer-source-v6 (ip6_addr | *) port ip_port ;
     alt-transfer-source (ip4_addr | *) port ip_port ;
     alt-transfer-source-v6 (ip6_addr | *) port ip_port ;
     use-alt-transfer-source yes_or_no;
     notify-source (ip4_addr | *) port ip_port ;
     notify-source-v6 (ip6_addr | *) port ip_port ;
     zone-statistics yes_or_no ;
     database string ;
     min-refresh-time number ;
     max-refresh-time number ;
     min-retry-time number ;
```

```
     max-retry-time number ;
     multi-master yes_or_no ;
};

zone zone_name class {
    type hint;
     file string ;
     delegation-only yes_or_no ;
     check-names (warn|fail|ignore) ; // Not Implemented.
};

zone zone_name class {
    type stub;
     allow-query { address_match_list } ;
     check-names (warn|fail|ignore) ;
     dialup dialup_option ;
     delegation-only yes_or_no ;
     file string ;
     forward (only|first) ;
     forwarders {  ip_addr port ip_port ; ...  };
     masters port ip_port { ( masters_list | ip_addr port ip_port key key ) ; ... } ;
     max-transfer-idle-in number ;
     max-transfer-time-in number ;
     pubkey number number number string ;
     transfer-source (ip4_addr | *) port ip_port ;
     transfer-source-v6 (ip6_addr | *) port ip_port ;
     alt-transfer-source (ip4_addr | *) port ip_port ;
     alt-transfer-source-v6 (ip6_addr | *) port ip_port ;
     use-alt-transfer-source yes_or_no;
     zone-statistics yes_or_no ;
     sig-validity-interval number ;
     database string ;
     min-refresh-time number ;
     max-refresh-time number ;
     min-retry-time number ;
     max-retry-time number ;
     multi-master yes_or_no ;
};

zone zone_name class {
    type forward;
     forward (only|first) ;
     forwarders {  ip_addr port ip_port ; ...  };
     delegation-only yes_or_no ;
};

zone zone_name class {
    type delegation-only;
};
```

6.2.24 zone Statement Definition and Usage

6.2.24.1 Zone Types

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

`slave`	A slave zone is a replica of a master zone. The **masters** list specifies one or more IP addresses of master servers that the slave contacts to update its copy of the zone. Masters list elements can also be names of other masters lists. By default, transfers are made from port 53 on the servers; this can be changed for all servers by specifying a port number before the list of IP addresses, or on a per-server basis after the IP address. Authentication to the master can also be done with per-server TSIG keys. If a file is specified, then the replica will be written to this file whenever the zone is changed, and reloaded from this file on a server restart. Use of a file is recommended, since it often speeds server startup and eliminates a needless waste of bandwidth. Note that for large numbers (in the tens or hundreds of thousands) of zones per server, it is best to use a two-level naming scheme for zone file names. For example, a slave server for the zone `example.com` might place the zone contents into a file called `ex/example.com` where `ex/` is just the first two letters of the zone name. (Most operating systems behave very slowly if you put 100 000 files into a single directory.)
`stub`	A stub zone is similar to a slave zone, except that it replicates only the NS records of a master zone instead of the entire zone. Stub zones are not a standard part of the DNS; they are a feature specific to the BIND implementation. Stub zones can be used to eliminate the need for glue NS record in a parent zone at the expense of maintaining a stub zone entry and a set of name server addresses in `named.conf`. This usage is not recommended for new configurations, and BIND 9 supports it only in a limited way. In BIND 4/8, zone transfers of a parent zone included the NS records from stub children of that zone. This meant that, in some cases, users could get away with configuring child stubs only in the master server for the parent zone. BIND 9 never mixes together zone data from different zones in this way. Therefore, if a BIND 9 master serving a parent zone has child stub zones configured, all the slave servers for the parent zone also need to have the same child stub zones configured. Stub zones can also be used as a way of forcing the resolution of a given domain to use a particular set of authoritative servers. For example, the caching name servers on a private network using RFC1918 addressing may be configured with stub zones for `10.in-addr.arpa` to use a set of internal name servers as the authoritative servers for that domain.
`forward`	A "forward zone" is a way to configure forwarding on a per-domain basis. A **zone** statement of type **forward** can contain a **forward** and/or **forwarders** statement, which will apply to queries within the domain given by the zone name. If no **forwarders** statement is present or an empty list for **forwarders** is given, then no forwarding will be done for the domain, canceling the effects of any forwarders in the **options** statement. Thus if you want to use this type of zone to change the behavior of the global **forward** option (that is, "forward first" to, then "forward only", or vice versa, but want to use the same servers as set globally) you need to re-specify the global forwarders.
`hint`	The initial set of root name servers is specified using a "hint zone". When the server starts up, it uses the root hints to find a root name server and get the most recent list of root name servers. If no hint zone is specified for class IN, the server uses a compiled-in default set of root servers hints. Classes other than IN have no built-in defaults hints.

`delegation-only`	This is used to enforce the delegation-only status of infrastructure zones (e.g. COM, NET, ORG). Any answer that is received without an explicit or implicit delegation in the authority section will be treated as NXDOMAIN. This does not apply to the zone apex. This should not be applied to leaf zones. `delegation-only` has no effect on answers received from forwarders.

6.2.24.2 Class

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

The `hesiod` class is named 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. The keyword `HS` is a synonym for hesiod.

Another MIT development is CHAOSnet, a LAN protocol created in the mid-1970s. Zone data for it can be specified with the `CHAOS` class.

6.2.24.3 Zone Options

allow-notify See the description of **allow-notify** in Section 6.2.16.4.

allow-query See the description of **allow-query** in Section 6.2.16.4.

allow-transfer See the description of **allow-transfer** in Section 6.2.16.4.

allow-update Specifies which hosts are allowed to submit Dynamic DNS updates for master zones. The default is to deny updates from all hosts. Note that allowing updates based on the requestor's IP address is insecure; see Section 7.3 for details.

update-policy Specifies a "Simple Secure Update" policy. See Section 6.2.24.4.

allow-update-forwarding See the description of **allow-update-forwarding** in Section 6.2.16.4.

also-notify Only meaningful if **notify** 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 name servers (other than the primary master) for the zone plus any IP addresses specified with **also-notify**. A port may be specified with each **also-notify** address to send the notify messages to a port other than the default of 53. **also-notify** is not meaningful for stub zones. The default is the empty list.

check-names This option is used to restrict the character set and syntax of certain domain names in master files and/or DNS responses received from the network. The default varies according to zone type. For **master** zones the default is **fail**. For **slave** zones the default is **warn**.

database Specify the type of database to be used for storing the zone data. The string following the **database** keyword is interpreted as a list of whitespace-delimited words. The first word identifies the database type, and any subsequent words are passed as arguments to the database to be interpreted in a way specific to the database type.

The default is **"rbt"**, BIND 9's native in-memory red-black-tree database. This database does not take arguments.

Other values are possible if additional database drivers have been linked into the server. Some sample drivers are included with the distribution but none are linked in by default.

dialup See the description of **dialup** in Section 6.2.16.1.

delegation-only The flag only applies to hint and stub zones. If set to **yes**, then the zone will also be treated as if it is also a delegation-only type zone.

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

forwarders Used to override the list of global forwarders. If it is not specified in a zone of type **forward**, no forwarding is done for the zone and the global options are not used.

ixfr-base Was used in BIND 8 to specify the name of the transaction log (journal) file for dynamic update and IXFR. BIND 9 ignores the option and constructs the name of the journal file by appending "`.jnl`" to the name of the zone file.

ixfr-tmp-file Was an undocumented option in BIND 8. Ignored in BIND 9.

max-transfer-time-in See the description of **max-transfer-time-in** in Section 6.2.16.7.

max-transfer-idle-in See the description of **max-transfer-idle-in** in Section 6.2.16.7.

max-transfer-time-out See the description of **max-transfer-time-out** in Section 6.2.16.7.

max-transfer-idle-out See the description of **max-transfer-idle-out** in Section 6.2.16.7.

notify See the description of **notify** in Section 6.2.16.1.

pubkey In BIND 8, this option was intended for specifying a public zone key for verification of signatures in DNSSEC signed zones when they are loaded from disk. BIND 9 does not verify signatures on load and ignores the option.

zone-statistics If **yes**, the server will keep statistical information for this zone, which can be dumped to the **statistics-file** defined in the server options.

sig-validity-interval See the description of **sig-validity-interval** in Section 6.2.16.15.

transfer-source See the description of **transfer-source** in Section 6.2.16.7.

transfer-source-v6 See the description of **transfer-source-v6** in Section 6.2.16.7.

alt-transfer-source See the description of **alt-transfer-source** in Section 6.2.16.7.

alt-transfer-source-v6 See the description of **alt-transfer-source-v6** in Section 6.2.16.7.

use-alt-transfer-source See the description of **use-alt-transfer-source** in Section 6.2.16.7.

notify-source See the description of **notify-source** in Section 6.2.16.7

notify-source-v6 See the description of **notify-source-v6** in Section 6.2.16.7.

min-refresh-time, max-refresh-time, min-retry-time, max-retry-time See the description in Section 6.2.16.15.

ixfr-from-differences See the description of **ixfr-from-differences** in Section 6.2.16.1.

key-directory See the description of **key-directory** in Section 6.2.16.

multi-master See the description of **multi-master** in Section 6.2.16.1.

6.2.24.4 Dynamic Update Policies

BIND 9 supports two alternative methods of granting clients the right to perform dynamic updates to a zone, configured by the **allow-update** and **update-policy** option, respectively.

The **allow-update** clause works the same way as in previous versions of BIND. It grants given clients the permission to update any record of any name in the zone.

The **update-policy** clause is new in BIND 9 and allows more fine-grained control over what updates are allowed. A set of rules is specified, where each rule either grants or denies permissions for one or more names to be updated by one or more identities. If the dynamic update request message is signed (that is, it includes either a TSIG or SIG(0) record), the identity of the signer can be determined.

Rules are specified in the **update-policy** zone option, and are only meaningful for master zones. When the **update-policy** statement is present, it is a configuration error for the **allow-update** statement to be present. The **update-policy** statement only examines the signer of a message; the source address is not relevant.

This is how a rule definition looks:

```
( grant | deny ) identity nametype name  types
```

Each rule grants or denies privileges. Once a message has successfully matched a rule, the operation is immediately granted or denied and no further rules are examined. A rule is matched when the signer matches the identity field, the name matches the name field in accordance with the nametype field, and the type matches the types specified in the type field.

The identity field specifies a name or a wildcard name. Normally, this is the name of the TSIG or SIG(0) key used to sign the update request. When a TKEY exchange has been used to create a shared secret, the identity of the shared secret is the same as the identity of the key used to authenticate the TKEY exchange. When the *`identity`* field specifies a wildcard name, it is subject to DNS wildcard expansion, so the rule will apply to multiple identities. The *`identity`* field must contain a fully qualified domain name.

The *`nametype`* field has 4 values: `name`, `subdomain`, `wildcard`, and `self`.

`name`	Exact-match semantics. This rule matches when the name being updated is identical to the contents of the *`name`* field.
`subdomain`	This rule matches when the name being updated is a subdomain of, or identical to, the contents of the *`name`* field.

`wildcard`	The *`name`* field is subject to DNS wildcard expansion, and this rule matches when the name being updated name is a valid expansion of the wildcard.
`self`	This rule matches when the name being updated matches the contents of the *`identity`* field. The *`name`* field is ignored, but should be the same as the *`identity`* field. The `self` nametype is most useful when allowing using one key per name to update, where the key has the same name as the name to be updated. The *`identity`* would be specified as `*` in this case.

In all cases, the *`name`* field must specify a fully qualified domain name.

If no types are explicitly specified, this rule matches all types except SIG, NS, SOA, and NXT. Types may be specified by name, including "ANY" (ANY matches all types except NXT, which can never be updated). Note that when an attempt is made to delete all records associated with a name, the rules are checked for each existing record type.

6.3 Zone File

6.3.1 Types of Resource Records and When to Use Them

This section, largely borrowed from RFC 1034, describes the concept of a Resource Record (RR) and explains when each is used. Since the publication of RFC 1034, several new RRs have been identified and implemented in the DNS. These are also included.

6.3.1.1 Resource Records

A domain name identifies a node. Each node has a set of resource information, which may be empty. The set of resource information associated with a particular name is composed of separate RRs. The order of RRs in a set is not significant and need not be preserved by name servers, resolvers, or other parts of the DNS. However, sorting of multiple RRs is permitted for optimization purposes, for example, to specify that a particular nearby server be tried first. See Section 6.2.16.13 and Section 6.2.16.14.

The components of a Resource Record are:

owner name	the domain name where the RR is found.
type	an encoded 16-bit value that specifies the type of the resource record.
TTL	the time-to-live of the RR. This field is a 32-bit integer in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded.
class	an encoded 16-bit value that identifies a protocol family or instance of a protocol.
RDATA	the resource data. The format of the data is type (and sometimes class) specific.

The following are *types* of valid RRs:

A	a host address. In the IN class, this is a 32-bit IP address. Described in RFC 1035.
AAAA	IPv6 address. Described in RFC 1886.

A6	IPv6 address. This can be a partial address (a suffix) and an indirection to the name where the rest of the address (the prefix) can be found. Experimental. Described in RFC 2874.
AFSDB	location of AFS database servers. Experimental. Described in RFC 1183.
APL	address prefix list. Experimental. Described in RFC 3123.
CERT	holds a digital certificate. Described in RFC 2538.
CNAME	identifies the canonical name of an alias. Described in RFC 1035.
DNAME	Replaces the domain name specified with another name to be looked up, effectively aliasing an entire subtree of the domain name space rather than a single record as in the case of the CNAME RR. Described in RFC 2672.
GPOS	Specifies the global position. Superseded by LOC.
HINFO	identifies the CPU and OS used by a host. Described in RFC 1035.
ISDN	representation of ISDN addresses. Experimental. Described in RFC 1183.
KEY	stores a public key associated with a DNS name. Described in RFC 2535.
KX	identifies a key exchanger for this DNS name. Described in RFC 2230.
LOC	for storing GPS info. Described in RFC 1876. Experimental.
MX	identifies a mail exchange for the domain. A 16-bit preference value (lower is better) followed by the host name of the mail exchange. Described in RFC 974, RFC 1035.
NAPTR	name authority pointer. Described in RFC 2915.
NSAP	a network service access point. Described in RFC 1706.
NS	the authoritative name server for the domain. Described in RFC 1035.
NXT	used in DNSSEC to securely indicate that RRs with an owner name in a certain name interval do not exist in a zone and indicate what RR types are present for an existing name. Described in RFC 2535.
PTR	a pointer to another part of the domain name space. Described in RFC 1035.
PX	provides mappings between RFC 822 and X.400 addresses. Described in RFC 2163.
RP	information on persons responsible for the domain. Experimental. Described in RFC 1183.
RT	route-through binding for hosts that do not have their own direct wide area network addresses. Experimental. Described in RFC 1183.
SIG	(”signature”) contains data authenticated in the secure DNS. Described in RFC 2535.
SOA	identifies the start of a zone of authority. Described in RFC 1035.
SRV	information about well known network services (replaces WKS). Described in RFC 2782.
TXT	text records. Described in RFC 1035.
WKS	information about which well known network services, such as SMTP, that a domain supports. Historical.
X25	representation of X.25 network addresses. Experimental. Described in RFC 1183.

The following *classes* of resource records are currently valid in the DNS:

IN	The Internet.
CH	CHAOSnet, a LAN protocol created at MIT in the mid-1970s. Rarely used for its historical purpose, but reused for BIND's built-in server information zones, e.g., `version.bind`.
HS	Hesiod, an information service developed by MIT's Project Athena. It is used to share information about various systems databases, such as users, groups, printers and so on.

The owner name is often implicit, rather than forming an integral part of the RR. For example, many name servers internally form tree or hash structures for the name space, and chain RRs off nodes. The remaining RR parts are the fixed header (type, class, TTL) which is consistent for all RRs, and a variable part (RDATA) that fits the needs of the resource being described.

The meaning of the TTL field is a time limit on how long an RR can be kept in a cache. This limit does not apply to authoritative data in zones; it is also timed out, but by the refreshing policies for the zone. The TTL is assigned by the administrator for the zone where the data originates. While short TTLs can be used to minimize caching, and a zero TTL prohibits caching, the realities of Internet performance suggest that these times should be on the order of days for the typical host. If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency during the change, and then increased back to its former value following the change.

The data in the RDATA section of RRs is carried as a combination of binary strings and domain names. The domain names are frequently used as "pointers" to other data in the DNS.

6.3.1.2 Textual expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In the examples provided in RFC 1034, a style similar to that used in master files was employed in order to show the contents of RRs. In this format, most RRs are shown on a single line, although continuation lines are possible using parentheses.

The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

Following the owner, we list the TTL, type, and class of the RR. Class and type use the mnemonics defined above, and TTL is an integer before the type field. In order to avoid ambiguity in parsing, type and class mnemonics are disjoint, TTLs are integers, and the type mnemonic is always last. The IN class and TTL values are often omitted from examples in the interests of clarity.

The resource data or RDATA section of the RR are given using knowledge of the typical representation for the data.

For example, we might show the RRs carried in a message as:

```
ISI.EDU.            MX          10 VENERA.ISI.EDU.
                    MX          10 VAXA.ISI.EDU
VENERA.ISI.EDU      A           128.9.0.32
                    A           10.1.0.52
VAXA.ISI.EDU        A           10.2.0.27
                    A           128.9.0.33
```

The MX RRs have an RDATA section which consists of a 16-bit number followed by a domain name. The address RRs use a standard IP address format to contain a 32-bit internet address.

The above example shows six RRs, with two RRs at each of three domain names.

Similarly we might see:

```
XX.LCS.MIT.EDU.      A              10.0.0.44
IN
CH                   A              MIT.EDU. 2420
```

This example shows two addresses for `XX.LCS.MIT.EDU`, each of a different class.

6.3.2 Discussion of MX Records

As described above, domain servers store information as a series of resource records, each of which contains a particular piece of information about a given domain name (which is usually, but not always, a host). The simplest way to think of a RR is as a typed pair of data, a domain name matched with a relevant datum, and stored with some additional type information to help systems determine when the RR is relevant.

MX records are used to control delivery of email. The data specified in the record is a priority and a domain name. The priority controls the order in which email delivery is attempted, with the lowest number first. If two priorities are the same, a server is chosen randomly. If no servers at a given priority are responding, the mail transport agent will fall back to the next largest priority. Priority numbers do not have any absolute meaning — they are relevant only respective to other MX records for that domain name. The domain name given is the machine to which the mail will be delivered. It *must* have an associated A record — CNAME is not sufficient.

For a given domain, if there is both a CNAME record and an MX record, the MX record is in error, and will be ignored. Instead, the mail will be delivered to the server specified in the MX record pointed to by the CNAME.

```
example.com.            IN      MX      10              mail.example.com.
                        IN      MX      10              mail2.example.com.
                        IN      MX      20              mail.backup.org.
mail.example.com.       IN      A       10.0.0.1
mail2.example.com.      IN      A       10.0.0.2
```

For example:

Mail delivery will be attempted to `mail.example.com` and `mail2.example.com` (in any order), and if neither of those succeed, delivery to `mail.backup.org` will be attempted.

6.3.3 Setting TTLs

The time-to-live of the RR field is a 32-bit integer represented in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded. The following three types of TTL are currently used in a zone file.

SOA	The last field in the SOA is the negative caching TTL. This controls how long other servers will cache no-such-domain (NXDOMAIN) responses from you. The maximum time for negative caching is 3 hours (3h).
$TTL	The $TTL directive at the top of the zone file (before the SOA) gives a default TTL for every RR without a specific TTL set.
RR TTLs	Each RR can have a TTL as the second field in the RR, which will control how long other servers can cache the it.

All of these TTLs default to units of seconds, though units can be explicitly specified, for example, `1h30m`.

6.3.4 Inverse Mapping in IPv4

Reverse name resolution (that is, translation from IP address to name) is achieved by means of the *in-addr.arpa* domain and PTR records. Entries in the in-addr.arpa domain are made in least-to-most significant order, read left to right. This is the opposite order to the way IP addresses are usually written. Thus, a machine with an IP address of 10.1.2.3 would have a corresponding in-addr.arpa name of 3.2.1.10.in-addr.arpa. This name should have a PTR resource record whose data field is the name of the machine or, optionally, multiple PTR records if the machine has more than one name. For example, in the [example.com] domain:

```
$ORIGIN         2.1.10.in-addr.arpa
3               IN PTR foo.example.com.
```

NOTE

The **$ORIGIN** lines in the examples are for providing context to the examples only-they do not necessarily appear in the actual usage. They are only used here to indicate that the example is relative to the listed origin.

6.3.5 Other Zone File Directives

The Master File Format was initially defined in RFC 1035 and has subsequently been extended. While the Master File Format itself is class independent all records in a Master File must be of the same class.

Master File Directives include **$ORIGIN**, **$INCLUDE**, and **$TTL.**

6.3.5.1 The $ORIGIN Directive

Syntax: **$ORIGIN***domain-name* [*comment*]

$ORIGIN sets the domain name that will be appended to any unqualified records. When a zone is first read in there is an implicit **$ORIGIN** <zone-name>**.** The current **$ORIGIN** is appended to the domain specified in the **$ORIGIN** argument if it is not absolute.

```
$ORIGIN example.com.
WWW     CNAME   MAIN-SERVER
```

is equivalent to

```
WWW.EXAMPLE.COM. CNAME MAIN-SERVER.EXAMPLE.COM.
```

6.3.5.2 The $INCLUDE Directive

Syntax: **$INCLUDE** *filename* [*origin*] [*comment*]

Read and process the file `filename` as if it were included into the file at this point. If **origin** is specified the file is processed with **$ORIGIN** set to that value, otherwise the current **$ORIGIN** is used.

The origin and the current domain name revert to the values they had prior to the **$INCLUDE** once the file has been read.

NOTE

RFC 1035 specifies that the current origin should be restored after an **$INCLUDE**, but it is silent on whether the current domain name should also be restored. BIND 9 restores both of them. This could be construed as a deviation from RFC 1035, a feature, or both.

6.3.5.3 The $TTL Directive

Syntax: **$TTL** *default-ttl* [*comment*]

Set the default Time To Live (TTL) for subsequent records with undefined TTLs. Valid TTLs are of the range 0-2147483647 seconds.

$TTL is defined in RFC 2308.

6.3.6 BIND Master File Extension: the $GENERATE Directive

Syntax: **$GENERATE** *range lhs* [*ttl*] [*class*] *type rhs* [*comment*]

$GENERATE is used to create a series of resource records that only differ from each other by an iterator. **$GENERATE** can be used to easily generate the sets of records required to support sub /24 reverse delegations described in RFC 2317: Classless IN-ADDR.ARPA delegation.

```
$ORIGIN 0.0.192.IN-ADDR.ARPA.
$GENERATE 1-2 0 NS SERVER$.EXAMPLE.
$GENERATE 1-127 $ CNAME $.0
```

is equivalent to

```
0.0.0.192.IN-ADDR.ARPA NS SERVER1.EXAMPLE.
0.0.0.192.IN-ADDR.ARPA. NS SERVER2.EXAMPLE.
1.0.0.192.IN-ADDR.ARPA. CNAME 1.0.0.0.192.IN-ADDR.ARPA.
2.0.0.192.IN-ADDR.ARPA. CNAME 2.0.0.0.192.IN-ADDR.ARPA.
...
127.0.0.192.IN-ADDR.ARPA. CNAME 127.0.0.0.192.IN-ADDR.ARPA.
```

range	This can be one of two forms: start-stop or start-stop/step. If the first form is used, then step is set to 1. All of start, stop and step must be positive.
lhs	**lhs** describes the owner name of the resource records to be created. Any single **$** (dollar sign) symbols within the **lhs** side are replaced by the iterator value. To get a $ in the output you need to escape the **$** using a backslash \, e.g. **$**. The **$** may optionally be followed by modifiers which change the offset from the iterator, field width and base. Modifiers are introduced by a { immediately following the **$** as **$**{**offset[,width[,base]]**}. For example, **$**{**-20,3,d**} which subtracts 20 from the current value, prints the result as a decimal in a zero-padded field of width 3. Available output forms are decimal (**d**), octal (**o**) and hexadecimal (**x** or **X** for uppercase). The default modifier is **$**{**0,0,d**}. If the **lhs** is not absolute, the current **$ORIGIN** is appended to the name. For compatibility with earlier versions, **$$** is still recognized as indicating a literal $ in the output.

ttl	Specifies the ttl of the generated records. If not specified this will be inherited using the normal ttl inheritance rules. **class** and **ttl** can be entered in either order.
class	Specifies the class of the generated records. This must match the zone class if it is specified. **class** and **ttl** can be entered in either order.
type	At present the only supported types are PTR, CNAME, DNAME, A, AAAA and NS.
rhs	A domain name. It is processed similarly to lhs.

The **$GENERATE** directive is a BIND extension and not part of the standard zone file format.

BIND 8 does not support the optional TTL and CLASS fields.

Chapter 7

BIND 9 Security Considerations

7.1 Access Control Lists

Access Control Lists (ACLs), are address match lists that you can set up and nickname for future use in **allow-notify**, **allow-query**, **allow-recursion**, **blackhole**, **allow-transfer**, etc.

Using ACLs allows you to have finer control over who can access your name server, without cluttering up your config files with huge lists of IP addresses.

It is a *good idea* to use ACLs, and to control access to your server. Limiting access to your server by outside parties can help prevent spoofing and denial of service (DoS) attacks against your server.

Here is an example of how to properly apply ACLs:

```
// Set up an ACL named "bogusnets" that will block RFC1918 space,
// which is commonly used in spoofing attacks.
acl bogusnets { 0.0.0.0/8; 1.0.0.0/8; 2.0.0.0/8; 192.0.2.0/24; 224.0.0.0/3; 10.0.0.0/8

// Set up an ACL called our-nets. Replace this with the real IP numbers.
acl our-nets { x.x.x.x/24; x.x.x.x/21; };
options {
  ...
  ...
  allow-query { our-nets; };
  allow-recursion { our-nets; };
  ...
  blackhole { bogusnets; };
  ...
};

zone "example.com" {
  type master;
  file "m/example.com";
  allow-query { any; };
};
```

This allows recursive queries of the server from the outside unless recursion has been previously disabled.

For more information on how to use ACLs to protect your server, see the *AUSCERT* advisory at <ftp://ftp.auscert.org.au/pub/auscert/advisory/AL-1999.004.dns_dos>

7.2 chroot and setuid (for UNIX servers)

On UNIX servers, it is possible to run BIND in a *chrooted* environment (using the **chroot()** function) by specifying the "`-t`" option. This can help improve system security by placing BIND in a "sandbox", which will limit the damage done if a server is compromised.

Another useful feature in the UNIX version of BIND is the ability to run the daemon as an unprivileged user (`-u` *`user`*). We suggest running as an unprivileged user when using the **chroot** feature.

Here is an example command line to load BIND in a **chroot** sandbox, **/var/named**, and to run **named setuid** to user 202:

```
/usr/local/bin/named -u 202 -t /var/named
```

7.2.1 The chroot Environment

In order for a **chroot** environment to work properly in a particular directory (for example, `/var/named`), you will need to set up an environment that includes everything BIND needs to run. From BIND's point of view, `/var/named` is the root of the filesystem. You will need to adjust the values of options like like **directory** and **pid-file** to account for this.

Unlike with earlier versions of BIND, you will typically *not* need to compile **named** statically nor install shared libraries under the new root. However, depending on your operating system, you may need to set up things like `/dev/zero`, `/dev/random`, `/dev/log`, and `/etc/localtime`.

7.2.2 Using the setuid Function

Prior to running the **named** daemon, use the **touch** utility (to change file access and modification times) or the **chown** utility (to set the user id and/or group id) on files to which you want BIND to write. Note that if the **named** daemon is running as an unprivileged user, it will not be able to bind to new restricted ports if the server is reloaded.

7.3 Dynamic Update Security

Access to the dynamic update facility should be strictly limited. In earlier versions of BIND, the only way to do this was based on the IP address of the host requesting the update, by listing an IP address or network prefix in the **allow-update** zone option. This method is insecure since the source address of the update UDP packet is easily forged. Also note that if the IP addresses allowed by the **allow-update** option include the address of a slave server which performs forwarding of dynamic updates, the master can be trivially attacked by sending the update to the slave, which will forward it to the master with its own source IP address causing the master to approve it without question.

For these reasons, we strongly recommend that updates be cryptographically authenticated by means of transaction signatures (TSIG). That is, the **allow-update** option should list only TSIG key names, not IP addresses or network prefixes. Alternatively, the new **update-policy** option can be used.

Some sites choose to keep all dynamically-updated DNS data in a subdomain and delegate that subdomain to a separate zone. This way, the top-level zone containing critical data such as the IP addresses of public web and mail servers need not allow dynamic update at all.

Chapter 8

Troubleshooting

8.1 Common Problems

8.1.1 It's not working; how can I figure out what's wrong?

The best solution to solving installation and configuration issues is to take preventative measures by setting up logging files beforehand. The log files provide a source of hints and information that can be used to figure out what went wrong and how to fix the problem.

8.2 Incrementing and Changing the Serial Number

Zone serial numbers are just numbers-they aren't date related. A lot of people set them to a number that represents a date, usually of the form YYYYMMDDRR. A number of people have been testing these numbers for Y2K compliance and have set the number to the year 2000 to see if it will work. They then try to restore the old serial number. This will cause problems because serial numbers are used to indicate that a zone has been updated. If the serial number on the slave server is lower than the serial number on the master, the slave server will attempt to update its copy of the zone.

Setting the serial number to a lower number on the master server than the slave server means that the slave will not perform updates to its copy of the zone.

The solution to this is to add 2147483647 (2^31-1) to the number, reload the zone and make sure all slaves have updated to the new zone serial number, then reset the number to what you want it to be, and reload the zone again.

8.3 Where Can I Get Help?

The Internet Software Consortium (ISC) offers a wide range of support and service agreements for BIND and DHCP servers. Four levels of premium support are available and each level includes support for all ISC programs, significant discounts on products and training, and a recognized priority on bug fixes and non-funded feature requests. In addition, ISC offers a standard support agreement package which includes services ranging from bug fix announcements to remote support. It also includes training in BIND and DHCP.

To discuss arrangements for support, contact info@isc.org <mailto:info@isc.org> or visit the ISC web page at <http://www.isc.org/services/support/> to read more.

Appendix A

Appendices

A.1 Acknowledgments

A.1.1 A Brief History of the DNS and BIND

Although the "official" beginning of the Domain Name System occurred in 1984 with the publication of RFC 920, the core of the new system was described in 1983 in RFCs 882 and 883. From 1984 to 1987, the ARPAnet (the precursor to today's Internet) became a testbed of experimentation for developing the new naming/addressing scheme in a rapidly expanding, operational network environment. New RFCs were written and published in 1987 that modified the original documents to incorporate improvements based on the working model. RFC 1034, "Domain Names-Concepts and Facilities", and RFC 1035, "Domain Names-Implementation and Specification" were published and became the standards upon which all DNS implementations are built.

The first working domain name server, called "Jeeves", was written in 1983-84 by Paul Mockapetris for operation on DEC Tops-20 machines located at the University of Southern California's Information Sciences Institute (USC-ISI) and SRI International's Network Information Center (SRI-NIC). A DNS server for Unix machines, the Berkeley Internet Name Domain (BIND) package, was written soon after by a group of graduate students at the University of California at Berkeley under a grant from the US Defense Advanced Research Projects Administration (DARPA).

Versions of BIND through 4.8.3 were maintained by the Computer Systems Research Group (CSRG) at UC Berkeley. Douglas Terry, Mark Painter, David Riggle and Songnian Zhou made up the initial BIND project team. After that, additional work on the software package was done by Ralph Campbell. Kevin Dunlap, a Digital Equipment Corporation employee on loan to the CSRG, worked on BIND for 2 years, from 1985 to 1987. Many other people also contributed to BIND development during that time: Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss, Jim Bloom and Mike Schwartz. BIND maintenance was subsequently handled by Mike Karels and O. Kure.

BIND versions 4.9 and 4.9.1 were released by Digital Equipment Corporation (now Compaq Computer Corporation). Paul Vixie, then a DEC employee, became BIND's primary caretaker. He was assisted by Phil Almquist, Robert Elz, Alan Barrett, Paul Albitz, Bryan Beecher, Andrew Partan, Andy Cherenson, Tom Limoncelli, Berthold Paffrath, Fuat Baran, Anant Kumar, Art Harkin, Win Treese, Don Lewis, Christophe Wolfhugel, and others.

BIND version 4.9.2 was sponsored by Vixie Enterprises. Paul Vixie became BIND's principal architect/programmer.

BIND versions from 4.9.3 onward have been developed and maintained by the Internet Software Consortium with support being provided by ISC's sponsors. As co-architects/programmers, Bob Halley and Paul Vixie released the first production-ready version of BIND version 8 in May 1997.

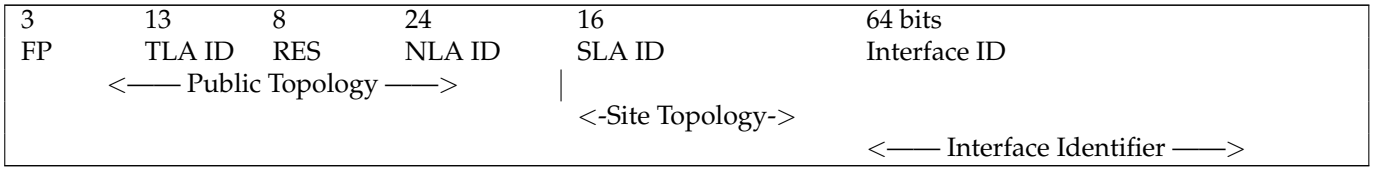
BIND development work is made possible today by the sponsorship of several corporations, and by the tireless work efforts of numerous individuals.

A.2 General DNS Reference Information

A.2.1 IPv6 addresses (AAAA)

IPv6 addresses are 128-bit identifiers for interfaces and sets of interfaces which were introduced in the DNS to facilitate scalable Internet routing. There are three types of addresses: *Unicast*, an identifier for a single interface; *Anycast*, an identifier for a set of interfaces; and *Multicast*, an identifier for a set of interfaces. Here we describe the global Unicast address scheme. For more information, see RFC 2374.

The aggregatable global Unicast address format is as follows:



Where

FP	=	Format Prefix (001)
TLA ID	=	Top-Level Aggregation Identifier
RES	=	Reserved for future use
NLA ID	=	Next-Level Aggregation Identifier
SLA ID	=	Site-Level Aggregation Identifier
INTERFACE ID	=	Interface Identifier

The *Public Topology* is provided by the upstream provider or ISP, and (roughly) corresponds to the IPv4 *network* section of the address range. The *Site Topology* is where you can subnet this space, much the same as subnetting an IPv4 /16 network into /24 subnets. The *Interface Identifier* is the address of an individual interface on a given network. (With IPv6, addresses belong to interfaces rather than machines.)

The subnetting capability of IPv6 is much more flexible than that of IPv4: subnetting can now be carried out on bit boundaries, in much the same way as Classless InterDomain Routing (CIDR).

The Interface Identifier must be unique on that network. On ethernet networks, one way to ensure this is to set the address to the first three bytes of the hardware address, "FFFE", then the last three bytes of the hardware address. The lowest significant bit of the first byte should then be complemented. Addresses are written as 32-bit blocks separated with a colon, and leading zeros of a block may be omitted, for example:

2001:db8:201:9:a00:20ff:fe81:2b32

IPv6 address specifications are likely to contain long strings of zeros, so the architects have included a shorthand for specifying them. The double colon ('::') indicates the longest possible string of zeros that can fit, and can be used only once in an address.

A.3 Bibliography (and Suggested Reading)

A.3.1 Request for Comments (RFCs)

Specification documents for the Internet protocol suite, including the DNS, are published as part of the Request for Comments (RFCs) series of technical notes. The standards themselves are defined by the Internet Engineering Task Force (IETF) and the Internet Engineering Steering Group (IESG). RFCs can be obtained online via FTP at ftp://www.isi.edu/in-notes/RFC*xxx*.txt <ftp://www.isi.edu/

in-notes/> (where *xxx* is the number of the RFC). RFCs are also available via the Web at <http://www.ietf.org/rfc/>.

References

Standards

[RFC1034] *Domain Names — Concepts and Facilities*, P.V. Mockapetris, November 1987.

[RFC1035] *Domain Names — Implementation and Specification*, P. V. Mockapetris, November 1987.

[RFC974] *Mail Routing and the Domain System*, C. Partridge, January 1986.

Proposed Standards

[RFC1995] *Incremental Zone Transfer in DNS*, M. Ohta, August 1996.

[RFC1996] *A Mechanism for Prompt Notification of Zone Changes*, P. Vixie, August 1996.

[RFC2136] *Dynamic Updates in the Domain Name System*, P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, April 1997.

[RFC2181] *Clarifications to the DNS Specification*, R., R. Bush Elz, July 1997.

[RFC2308] *Negative Caching of DNS Queries*, M. Andrews, March 1998.

[RFC2845] *Secret Key Transaction Authentication for DNS (TSIG)*, P. Vixie, O. Gudmundsson, D. Eastlake, 3rd, and B. Wellington, May 2000.

Proposed Standards Still Under Development

[RFC1886] *DNS Extensions to support IP version 6*, S. Thomson and C. Huitema, December 1995.

[RFC2065] *Domain Name System Security Extensions*, D. Eastlake, 3rd and C. Kaufman, January 1997.

[RFC2137] *Secure Domain Name System Dynamic Update*, D. Eastlake, 3rd, April 1997.

Other Important RFCs About DNS Implementation

[RFC1535] *A Security Problem and Proposed Correction With Widely Deployed DNS Software.*, E. Gavron, October 1993.

[RFC1536] *Common DNS Implementation Errors and Suggested Fixes*, A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller, October 1993.

[RFC1982] *Serial Number Arithmetic*, R. Elz and R. Bush, August 1996.

Resource Record Types

[RFC1183] *New DNS RR Definitions*, C.F. Everhart, L. A. Mamakos, R. Ullmann, and P. Mockapetris, October 1990.

[RFC1706] *DNS NSAP Resource Records*, B. Manning and R. Colella, October 1994.

[RFC1876] *A Means for Expressing Location Information in the Domain Name System*, C. Davis, P. Vixie, T., and I. Dickinson, January 1996.

[RFC2052] *A DNS RR for Specifying the Location of Services.*, A. Gulbrandsen and P. Vixie, October 1996.

[RFC2163] *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping*, A. Allocchio, January 1998.

[RFC2168] *Resolution of Uniform Resource Identifiers using the Domain Name System*, R. Daniel and M. Mealling, June 1997.

[RFC2230] *Key Exchange Delegation Record for the DNS*, R. Atkinson, October 1997.

DNS and the Internet

[RFC1101] *DNS Encoding of Network Names and Other Types*, P. V. Mockapetris, April 1989.

[RFC1123] *Requirements for Internet Hosts - Application and Support*, Braden, October 1989.

[RFC1591] *Domain Name System Structure and Delegation*, J. Postel, March 1994.

[RFC2317] *Classless IN-ADDR.ARPA Delegation*, H. Eidnes, G. de Groot, and P. Vixie, March 1998.

DNS Operations

[RFC1537] *Common DNS Data File Configuration Errors*, P. Beertema, October 1993.

[RFC1912] *Common DNS Operational and Configuration Errors*, D. Barr, February 1996.

[RFC2010] *Operational Criteria for Root Name Servers.*, B. Manning and P. Vixie, October 1996.

[RFC2219] *Use of DNS Aliases for Network Services.*, M. Hamilton and R. Wright, October 1997.

Other DNS-related RFCs

[RFC1464] *Using the Domain Name System To Store Arbitrary String Attributes*, R. Rosenbaum, May 1993.

[RFC1713] *Tools for DNS Debugging*, A. Romao, November 1994.

[RFC1794] *DNS Support for Load Balancing*, T. Brisco, April 1995.

[RFC2240] *A Legal Basis for Domain Name Allocation*, O. Vaughan, November 1997.

[RFC2345] *Domain Names and Company Name Retrieval*, J. Klensin, T. Wolf, and G. Oglesby, May 1998.

[RFC2352] *A Convention For Using Legal Names as Domain Names*, O. Vaughan, May 1998.

Obsolete and Unimplemented Experimental RRs

[RFC1712] *DNS Encoding of Geographical Location*, C. Farrell, M. Schulze, S. Pleitner, and D. Baldoni, November 1994.

A.3.2 Internet Drafts

Internet Drafts (IDs) are rough-draft working documents of the Internet Engineering Task Force. They are, in essence, RFCs in the preliminary stages of development. Implementors are cautioned not to regard IDs as archival, and they should not be quoted or cited in any formal documents unless accompanied by the disclaimer that they are "works in progress." IDs have a lifespan of six months after which they are deleted unless updated by their authors.

A.3.3 Other Documents About BIND

References

[1] *DNS and BIND*, Paul Albitz and Cricket Liu, Copyright © 1998 Sebastopol, CA: O'Reilly and Associates.