# BIND version 9

## ADMINISTRATOR REFERENCE MANUAL

# DRAFT

**March 22, 2000**



INTERNET SOFTWARE CONSORTIUM

# Section 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 an Internet nameserver 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.

## 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 BINDv9 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. The contents of *Section 5* are organized as in a reference manual to aid in the ongoing maintenance of the software. *Section 6* addresses security considerations, and *Section 7* contains troubleshooting help. The main body of the document is followed by several *Appendices* which contain useful reference information, such as a *Glossary* and a *Bibliography*, as well as historic information related to BIND and the Domain Name System.

## 1.3 Conventions Used in This Document

In this document, the following general typographic conventions are used:

| When describing: | Style Used: |
|---|---|
| A pathname, filename, URL, hostname, or mailing list name | *Times Italic* |
| A new term or concept | *Times Italic* |
| Literal user input | **`Courier Bold`** |
| Variable user input | *`Courier Italic`* |
| Program output | `Courier Plain` |

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

| When describing: | Style Used: |
|---|---|
| keywords | **Arial Bold** |
| variables | *Arial Italic* |

| | |
|---|---|
| "meta-syntactic" information (within brackets when optional) | *Courier Italic* |
| Command line input | **Courier Bold** |
| Program output | Courier Plain |
| Optional input | Text is enclosed in square brackets |

## 1.4 Discussion of Domain Name System (DNS) Basics and BIND

The purpose of this document is to explain the installation and basic upkeep of the BIND software package, and we begin by reviewing the fundamentals of the domain naming system as they relate to BIND. BIND consists of a *nameserver* (or "daemon") called **named** and a **resolver** library. The BIND server runs in the background, servicing queries on a well known network port. The standard port for UDP and TCP, usually port 53, is specified in **/etc/services**. The *resolver* is a set of routines residing in a system library that provides the interface that programs can use to access the domain name services.

### 1.4.1 Nameservers

A nameserver (NS) is a program that stores information about named resources and responds to queries from programs called *resolvers* which act as client processes. The basic function of an NS is to provide information about network objects by answering queries.

With the nameserver, the network can be broken into a hierarchy of domains. The name space is organized as a tree according to organizational or administrative boundaries. Each node of the tree, called a domain, is given a label. The name of the domain is the concatenation of all the labels of the domains from the root to the current domain. 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 domain. The whole name space is partitioned into areas called *zones*, each starting at a domain and extending down to the leaf domains or to domains where other zones start. Zones usually represent administrative boundaries. For example, a domain name for a host at the company *Example, Inc.* would be:

*ourhost.example.com*

The top level domain for corporate organizations is *com*; *example* is a subdomain of .*com*; and *ourhost* is the name of the host.

The specifications for the domain nameserver are defined in RFC1034, RFC1035 and RFC974. These documents can be found in */usr/src/etc/named/doc* in 4.4BSD or are available via **FTP** from *ftp://www.isi.edu/in-notes/* or via the Web at *http://www.ietf.org/rfc/*. (See Appendix C for complete information on finding and retrieving RFCs.) It is also recommended that you read the related **man** pages: **named** and **resolver**.

### 1.4.2 Types of Zones

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 domain 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 has one or more NS records in the parent zone, which should be matched by equivalent NS records at the root of the delegated zone (i.e., the "@" name in the zone file).

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

As an example, 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 nameservers. 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 it is suggested that you read RFCs 1033, 1034, and 1035 to gain a complete understanding of this difficult and subtle topic.

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

Each zone will have one *primary master* (also called *primary*) server which 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. There there will be some number of *secondary master* servers, which load the zone contents using the DNS protocol (that is, the secondary servers will contact the primary and fetch the zone data using TCP). This set of servers—the primary and all of its secondaries—should be listed in the NS records in the parent zone and will constitute a *delegation*. This set of servers must also be listed in the zone file itself, usually under the `@` name which indicates the *top level* or *root* of the current 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 in the zone's `@`.

Any servers listed in the NS records must be configured as authoritative for the zone. A server is authoritative for a zone when it has been configured to answer questions for that zone with authority, which it does by setting the "authoritative answer" (AA) bit in reply packets. A server may be authoritative for more than one zone. The authoritative data for a zone is composed of all of the Resource Records (RRs)—the data associated with names in a tree-structured name space—attached to all of the nodes from the top node of the zone down to leaf nodes or nodes above cuts around the bottom edge of the zone.

Adding a zone as a type master or type slave will tell the server to answer questions for the zone authoritatively. If the server is able to load the zone into memory without any errors it will set the AA bit when it replies to queries for the zone. See RFCs 1034 and 1035 for more information about the AA bit.

### 1.4.3 Servers

A DNS server can be master for some zones and slave for others or can be only a master, or only a slave, or can serve no zones and just answer queries via its *cache*. Master servers are often also called *primaries* and slave servers are often also called *secondaries*. Both master/primary and slave/secondary servers are authoritative for a zone.

All servers keep data in their cache until the data expires, based on a TTL (Time To Live) field which is maintained for all resource records.

#### 1.4.3.1 Master Server

The *primary master* server is the ultimate source of information about a domain. The primary master is an authoritative server configured to be the source of zone transfer for one or more secondary servers. The primary master server obtains data for the zone from a file on disk.

#### 1.4.3.2 Slave Server

A *slave server*, also called a *secondary server*, is an authoritative server that uses zone transfers from the primary master server to retrieve the zone data. Optionally, the slave server obtains zone data from a cache on disk. Slave servers provide necessary redundancy. All secondary/slave servers are named in the NS resource records (RRs) for the zone.

#### 1.4.3.3 Caching Only Server

Some servers are *caching only servers*. This means that the server caches the information that it receives and uses it until the data expires. A caching only server is a server that is not authoritative for any zone. This server services queries and asks other servers, who have the authority, for the information it needs.

#### 1.4.3.4 Forwarding Server

Instead of interacting with the nameservers for the root and other domains, a *forwarding server* always forwards queries it cannot satisfy from its authoritative data or cache to a fixed list of other servers. The forwarded queries are also known as *recursive queries*, the same type as a client would send to a server. There may be one or more servers forwarded to, and they are queried in turn until the list is exhausted or an answer is found. A forwarding server is typically used when you do not wish all the servers at a given site to interact with the rest of the Internet servers. A typical scenario would involve a number of internal DNS servers, and an internet firewall. The servers which cannot pass packets through the firewall would forward to the server which can, which would ask 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 workstations can take advantage of.

There is no prohibition against declaring a server to be a forwarder even though it has master and/or slave zones as well; the effect will still be that anything in the local server's cache or zones will be answered, and anything else will be forwarded using the forwarders list.

#### 1.4.3.5 Stealth Server

A stealth server is a server that answers authoritatively for a zone, but is not listed in that zone's NS records. Stealth servers can be used as a way to centralize distribution of a zone, without having to edit the zone on a remote nameserver. Where the master file for a zone resides on a stealth server in this way, it is often referred to as a 'hidden primary' configuration. Stealth servers can also be a way to keep a local copy of a zone for rapid access to the zone's records, even if all 'official' nameservers for the zone are inaccessable.

# Section 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 BINDv9 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. BINDv9 is now fully multithreaded, allowing full utilization of multiprocessor systems, for installations that need it.

## 2.2 CPU Requirements

CPU requirements for BINDv9 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. Future releases of BINDv9 will provide methods 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 nameserver 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. Ideally, the resource limits should be set higher than this stable size.

## 2.4 Nameserver Intensive Environment Issues

For nameserver intensive environments, there are two alternative configurations that may be used. The first is where clients and any second-level internal nameservers query a main nameserver, 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 nameservers 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 nameservers share their cached data.

## 2.5 Operating Systems Supported by the Internet Software Consortium

ISC BINDv9 compiles and runs on the following operating systems:

`IBM AIX 4.3`

`Compaq Digital/Tru64 UNIX 4.0D`

`HP HP-UX 11`

```
IRIX64 6.5
Red Hat Linux 6.0, 6.1
Sun Solaris 2.6, 7, 8 (beta)
FreeBSD 3.4-STABLE
NetBSD-current with "unproven" pthreads
```

# Section 3. Nameserver 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 Configuration and Logging

```
logging {
       channel named_log {
           file “logs/named.log”;
           print-time yes;
           print-category yes;
           print-severity yes;
           severity info;
};
       channel security_log {
           file “logs/security.log” versions 7 ;
           print-time yes;
};
       category default { named_log; default_debug; };
       category security { security_log };
};
// The two corporate subnets.  Use real IP numbers here in the real world.
acl corpnet { 192.168.4.0/24; 192.168.7.0/24; };
// The options statement.
options {
   directory “/etc/namedb”;    // Directory
   pid-file “named.pid”;        // Put .pid file in named directory.
   named-xfer “/path/to/named-xfer”;  // Where is our named-xfer ?
   check-names master fail;    // Fail on db errors in master zones.
   check-names slave warn;     // Warn about db errors
                              // in slave zones.
   check-names response warn;  // Warn about invalid responses
   use-id-pool yes;            // Help prevent spoofing
   host-statistics yes;        // Keep track of hosts/servers
                               // we’ve talked to.
   listen-on { 192.168.7.20; };        // Listen on this address.
   query-source address 192.168.7.20 port 53 ;
                                     // Source queries from port 53
                                      // to get past firewall.
   allow-transfer { none; };        // Don’t allow anyone to
                                      // transfer zones.
   allow-query { corpnet; };        // Allow only corpnets to query server.
                                      // Helps prevent DoS, spoofing.
   allow-recursion { corpnet; };     // Same, except this is for recursion.
};

include “keys.conf”;                        // Include a keys.conf with
                                            // TSIG/DNSSEC keys.
                                            // Shouldn’t be readable to anyone
                                             // except BIND user.
zone “.”{ type hint; file “local/named.root”; };
                                             // root hints

zone “0.0.127.IN-ADDR.ARPA” {
```

```
        type master; file "local/localhost.db"; notify no;
                                        // localhost
};

zone "example.com" {            // Example zone for "example.com".
type master;                    // It's a master zone.
file "m/example.com.db";        // The file is here.
allow-query { any; };           // Allow anyone to query.
allow-transfer { corpnet; };    // Only allow corp nets to transfer zone.
};

zone "offsite.example.com" {     // Example zone for an off-site corp zone.
type slave;                      // It's a slave zone.
masters { 192.168.4.12; };       // The master is at this address.
file "s/offsite.example.com.db"; // The file is here.
notify no;                       // Don't worry about NOTIFYing.
allow-query { any; };            // Allow anyone to query.
;
```

## 3.2 Load Balancing and Round Robin

Primitive load balancing can be achieved in DNS 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 record like the following means that clients will connect to each machine one third of the time:

| Name | TTL | CLASS | TYPE | Resource Record (RR) Data |
|---|---|---|---|---|
| www | 10m | IN | A | 10.0.0.1 |
| | 10m | IN | A | 10.0.0.2 |
| | 10m | 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. This is known as cyclic or round-robin ordering.In the example above, the first client will receive the records in the order 1,2,3; the second client will receive them in the order 2,3,1; and the third 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 in "RRset Ordering" on page 45.

## 3.3 Notify

DNS Notify is a mechanism that allows master nameservers to notify their slave servers of changes to a zone's data and that a query should be initiated to discover the new data. DNS Notify is turned on by default.

DNS Notify is fully documented in RFC 1996. See also the description of the zone option **`also-notify`** in section 3.1.3.7, "Zone transfers."

## 3.4 Nameserver Operations

### 3.4.1 Tools for Use With the Nameserver Daemon

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

#### 3.4.1.1 Diagnostic Tools

**dig**

The domain information groper (**`dig`**) is a command line tool that can be used to gather information from the Domain Name System servers. Dig 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 provides a simple DNS lookup using a command-line interface for looking up Internet hostnames. By default, the utility converts between host names and Internet addresses, but its functionality can be extended with the use of options.

**Usage**

```
host [-l] [-v] [-w] [-r] [-d] [-t querytype] [-a] host [server]
```

**nslookup**

**`nslookup`** is a program used to query Internet domain nameservers. nslookup has two modes: interactive and non-interactive. Interactive mode allows the user to query nameservers 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 nameserver will be used) or when the first argument is a hyphen (-) and the second argument is the host name or Internet address of a nameserver.

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 nameserver.

The options listed under the “set” command (see the nslookup man page for details) can be specified in the .nslookuprc file in the user’s home directory if they are listed one per line. Options can also be specified on the command line if they precede the arguments and are prefixed with a hyphen. For example, to change the default query type to host information, and the initial time-out to 10 seconds, type:

```
nslookup -query=hinfo -timeout=10
```

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

#### 3.4.1.2 Administrative Tools

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

**rndc**

The remote name daemon control (**`rndc`**) program is a program that allows the system administrator to control the operation of a nameserver. If you run rndc without any options it will display a usage message.

Usage:

```
rndc [-p port] [-m] server command [command ...]
```

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

#### 3.4.1.3 Monitoring Tools

**MRTG**

**`MRTG`** is primarily a router traffic grapher, but can be used to monitor BIND DNS servers, as well. The ‘stat’ script, supplied with MRTG in the MRTG ‘contrib/stat’ directory, can be used to monitor numbers of queries, and counts of various sorts of responses.

Return to the BINDv9 Administrator Reference Manual Table of Contents.

# Section 4. Advanced Concepts

## 4.1 Dynamic Update

Dynamic update is the term used for the ability under certain specified conditions to add, modify or delete records or RRsets in the master zone files. Dynamic update is fully described 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) works as specified in the *simple-secure-update* proposal. SIG and NXT 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.

The zone files of dynamic zones must not be edited by hand. The zone file on disk at any given time may not contain the latest changes performed by dynamic update. The zone file is written to disk only periodically, and changes that have occurred since the zone file was last written to disk are stored only in the zone's journal (*.jnl*) file. BIND 9 currently does not update the zone file when it exits like BIND 8 does, so editing the zone file manually is unsafe even when the server has been shut down.

## 4.2 Incremental Zone Transfers (IXFR)

The incremental zone transfer protocol (IXFR, RFC1995—see the list of "Proposed Standards" on page 79) is a way for slave servers to transfer only changed data, instead of having to transfer the entire zone every time it changes.

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, but not manually maintained master zones nor slave zones obtained by AXFR.

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.

## 4.3 Split DNS

Setting up different views, or visibility, of DNS space to internal , as opposed to external, resolvers is usually referred to as a "Split DNS" or "Split Brain DNS" setup. There are several reasons an organization would want to set its DNS up 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 e-mail 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 RFC1918 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 IP space and an external DMZ (the demilitarized zone, 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 nameservers. 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.example*, *site2.example*, *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 nameservers 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*, *ftp.example.com*), and mail exchanger 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 nameservers 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 nameservers 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 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; see "Sample Configuration and Logging" on page 9 for information on how to configure your zone files.

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 { bastion-ips-go-here; };// forward to external servers
    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 slave 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” {
  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
```

## 4.4 TSIG

Information about TSIG in this section was provided by Brian Wellington of TISLabs. This is a short guide to setting up 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-server communication. This includes zone transfer, notify, and recursive query messages. The resolver bundled with BIND 8.2 has limited support for TSIG, but it is doubtful that support will be integrated into any client applications.

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. Key-based access control is far superior (see *draft-ietf-dnsext-simple-secure-update-00.txt* in "Internet Drafts" on page 81). The **nsupdate** program that is shipped with BIND 8 supports TSIG via the "**-k**" command line option.

## 4.4.1 Generate Shared Keys for Each Pair of Hosts

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

### 4.4.1.1 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.

```
src/bin/dnskeygen/dnskeygen -H 128 -h -n host1-host2.
```

The key is in the file "Khost1-host2.+157+00000.private". Nothing actually uses this file, but the base64 encoded string following "Key:" can be extracted:

```
La/E5CjG9O+os1jq0a2jdA==
```

This string represents a shared secret.

### 4.4.1.2 Manual Generation

The shared secret is simply a random sequence of bits, encoded in base64. Most ASCII strings are valid base64 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 base64 encoded data.

## 4.4.2 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.

### 4.4.3 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's 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 succeeds, the response is signed by the same key.

### 4.4.4 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 host1's named.conf file, if host2's IP address 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 response 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 non-response messages to host1.

### 4.4.5 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.**"

### 4.4.6 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 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.

TSIG verification errors are logged by the server as
**"ns_req: TSIG verify failed - (reason)"**

which is printed at debug level 1.

## 4.5 DNSSEC

Cryptographc authentication of DNS information is made possible through the DNS Security (DNSSEC) extension to the domain system. This describes the processing of creating and using DNSSEC signed zones. The zones used in this exercise will be **dnssec.example** and **sub.dnssec.example**.

Step 1: Generate zone keys.

The following commands generate 640 bit DSA keys to be used as zone keys for the zones:

```
src/bin/dnskeygen/dnskeygen -D 640 -z -n dnssec.example.
src/bin/dnskeygen/dnskeygen -D 640 -z -n sub.dnssec.example.
```

In our example, keys with id 64555 and 39020 were generated.

Four files were created on disk:

**Kdnssec.example.+003+64555.key** (public key)

**Kdnssec.example.+003+64555.private** (private key)

**Ksub.dnssec.example.+003+39020.key** (public key)

**Ksub.dnssec.example.+003+39020.private** (private key)

The **.key** files contain public keys in DNS RR format, which is base 64. The **.private** files contain private keys, with each field encoded in base 64.

### Step 2: Enter the keys into the zones.

The parent zone needs its own key and the child key (as glue). The child zone needs its own key.

```
cat Kdnssec.example.+003+64555.key >> zone.dnssec.example
cat Ksub.dnssec.example.+003+39020.key >> zone.dnssec.example
cat Ksub.dnssec.example.+003+39020.key >> zone.sub.dnssec.example
```

Edit the zone files if desired (to move and/or format KEY records, etc.). This is also a good time to add **$ORIGIN** directives to the zone files if they aren't present.

### Step 3: Sign the parent zone.

The following command uses the zone.dnssec.example as input and creates the zone.dnssec.example.signed file. The key used is the dsa key for dnssec.example with id 64555 (**-ki**), and statistics are printed (**-st**). Parent files are generated for each child zone (**-ps**), and no global parent file is produced (**-no-p1**).

```
contrib/dns_signer/signer/dnssigner -zi zone.dnssec.example \
-zo zone.dnssec.example.signed -st -k1 dnssec.example dsa 64555 -ps
-no-p1
```

The following files are created:

**zone.dnssec.example.signed** (signed zone)

**sub.dnssec.example..PARENT** (parent file for sub.dnssec.example)

### Step 4: Sign the child zone.

The following command is similar to the previous one. The main difference is that the input parent file sub.dnssec.example..PARENT is specified (**-pi**) in addition to the input zone file; this file was generated by the previous call to the signer. Also, the -ps and -no-p1 options are omitted since there are no child zones of this zone. If this zone had child zones, these options should be present.

```
contrib/dns_signer/signer/dnssigner -zi zone.sub.dnssec.example \
-pi sub.dnssec.example..PARENT -zo zone.sub.dnssec.example.signed \
-st -k1 sub.dnssec.example dsa 39020
```

The following file is created:

**zone.sub.dnssec.example.signed** (signed zone)

### Step 5: Enter the top-level zone key in the named.conf file for the master server.

The public key for the top-level signed zone must be present in named.conf, so that the server can verify the data on load (it must be able to traverse a keychain and end at a trusted key). This key is added in a zone pubkey directive (which has a format similar to a KEY record, but not identical). Note that this is not needed for the subzone, as its key is signed by the trusted key in the parent zone.

This uses the key from Kdnssec.example.+003+64555.key

```
zone "dnssec.example" {
type master;
file "zone.dnssec.example.signed";
pubkey 16641 3 3 "AuNiWOmzSHwrzLMWv1C1gbKQBNAHwMeX+C0owQkfmdxjoTJvnmbN
       CdbGM/fnejQhEXsRT5l3NLy0H4UCX3ElGJT49n3nFb2jPuDYbkPh
       VV4sLfLJzQs/RWeQmQnNFF2HNmwksWlPvUT66k4mqJDtIk60Dio6
       1PML5sVDMQns7Zukq4aSn4jzRGkbDGhB9S3yzXVMVjYDwlM9frW9
       Ayt0vqDa0zG+V52YiCSOdFGWJ0bSFa8sTwcp4BEVUt/Kg2Zo4VAy
       +AeYLcQLb6vDZUX8x/BPByKKptfXirhNPv43xE6vT4xCxYPhvyDk
       Y7Qlf4W+/sSNNKE7P/JAKmQxxXAVPoXtBpa6";
};
```

Step 6: Enter the top-level zone key in the named.conf file for any other servers that will trust the key.

This uses the same key as above.

```
trusted-keys {
dnssec.example 16641 3 3
       "AuNiWOmzSHwrzLMWv1C1gbKQBNAHwMeX+C0owQkfmdxjoTJvnmbN
       CdbGM/fnejQhEXsRT5l3NLy0H4UCX3ElGJT49n3nFb2jPuDYbkPh
       VV4sLfLJzQs/RWeQmQnNFF2HNmwksWlPvUT66k4mqJDtIk60Dio6
       1PML5sVDMQns7Zukq4aSn4jzRGkbDGhB9S3yzXVMVjYDwlM9frW9
       Ayt0vqDa0zG+V52YiCSOdFGWJ0bSFa8sTwcp4BEVUt/Kg2Zo4VAy
       +AeYLcQLb6vDZUX8x/BPByKKptfXirhNPv43xE6vT4xCxYPhvyDk
       Y7Qlf4W+/sSNNKE7P/JAKmQxxXAVPoXtBpa6";
}
```

Start named.

## 4.6 IPv6

### 4.6.1 IPv6 addresses (A6)

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:

| 3 | 13 | 8 | 24 | 16 | 64 bits |
|---|---|---|---|---|---|
| FP | TLA ID | RES | NLA ID | SLA ID | Interface ID |

```
<------- Public Topology ------->
                                     <--Site Topology-->
                                                          <------ Interface Identifier ------>
```

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 like subnetting an IPv4 class A or B network into class Cs. 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 internal structure of the ‘Public Topology’ for an A6 global unicast address consists of:

| 3 | 13 | 8 | 24 |
|---|---|---|---|
| FP | TLA ID | RES | NLA ID |

A 3 bit FP (Format Prefix) of 001 indicates this is a global unicast address. FP lengths for other types of addresses may vary.

13 TLA (Top Level Aggregator) bits give the prefix of your top-level IP backbone carrier.

8 Reserved bits

24 bits for Next Level Aggregators. This allows organizations with a TLA to hand out portions of their IP space to client organizations, so that the client can then split up the network further by filling in more NLA bits, and hand out IPv6 prefixes to their clients, and so forth.

There is no particular structure for the ‘Site topology’ section. Organizations can allocate these bits in any way they desire, in the same way as they would subnet an IPv4 class A (8 bit prefix) network.

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:

**3ffe:8050:201:9: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.

### 4.6.2 Name to Address Lookup

Forward name lookups (host name to IP address) under IPv6 do not necessarily return the complete IPv6 address of the host. Because the provider-assigned prefix may change, the A6 record can simply specify the locally assigned portion of the name, and refer to the provider for the remainder.

A complete IPv6 A6 record that provides the full 128 bit address looks like:

```
$ORIGIN example.com.
; NAME               TTL TYPE     BITS IN REFERRAL   ADDRESS                              REFERRAL
host.example.com.    1h IN A6     0                  3ffe:8050:201:9:a00:20ff:fe81:2b32   .
```

Note that the number preceding the address is the number of bits to be provided via the referral. This is probably the easiest way to roll out an IPv6 installation, though you may wish to provide a reference to your provider assigned prefix:

```
$ORIGIN example.com.
; NAME                TTL TYPE     BITS IN REFERRAL   ADDRESS                  REFERRAL
host.example.com.     1h IN A6     48                 ::9:a00:20ff:fe81:2b32   prefix.example2.com.
```

Then, in example2.com's zone:

```
$ORIGIN example.com.
; NAME                  TTL TYPE     BITS IN REFERRAL   ADDRESS           REFERRAL
prefix.example2.com.    1h IN A6     0                  3ffe:8050:201::   .
```

The referral where there are no more bits is to '.', the root zone. Be warned that excessive use of this chaining can lead to extremely poor name resolution for people trying to access your hosts.

### 4.6.3 Address to Name Lookup

Reverse IPv6 addresses may appear as one or more hex strings, known as "bitstring labels," each followed by a number of valid bits. A full 128 bits may be specified at the ip6.int top level, or more likely, the provider will delegate you a smaller chunk of addresses for which you will need to supply reverse DNS.

The address can be split up along arbitrary boundaries, and is written with hex numbers in forward order, rather than in reverse order as IPv4 PTR records are written. The sections between dot separators are reversed as usual. If the number of valid bits in the hex string is less than the string specifies, it is the first N bits that are counted. Thus, \[x2/3] gives a bit pattern of 0010, the first three bits of which, 001, are valid.

The address above, then, is:

**`\[x3FFE8050020100090A0020FFFE812B32/128].ip6.int.`** (not divided)

**`\[x00090A0020FFFE812B32/80].\[xFFF402801008/45].\[x2/3].ip6.int.`** (divided into FP, TLA/RES/NLA, and local)

**`\[x00090A0020FFFE812B32/80].\[x80500201/32].\[xFFF0/13].\[x2/3].ip6.int.`** (divided into FP, TLA, RES/NLA, and local)

These strings are all equivalent. The combined TLA/RES/NLA in the second example bears no resemblance to any string in the address because it is offset by three bits.

### 4.6.4 Using DNAME for Delegation of IPv6 Reverse Addresses

Delegation of reverse addresses is done through the new DNAME RR. In the example above, where *\[x2/3].ip6.int.* needs to delegate *\[xFFF0]* to an organization (*example2.com*), the domain administrator would insert a line similar to the following in the *\[x2/3].ip6.int.* zone:

```
$ORIGIN \[x2/3].ip6.int.
\[xFFF0/13] 1h IN DNAME ip6.example2.com.
```

*example2.com* would then place into the *ip6* zone:

```
$ORIGIN ip6.example.com.
\[x80500201/32] 1h IN DNAME ip6.example.com.
```

Finally, *example.com* needs to include in the *ip6.example.com* zone:

```
$ORIGIN ip6.example.com.
\[x00090A0020FFFE812B32/80] 1h IN PTR host.example.com.
```

We suggest that the top of your administrative control (*example.com*, in this case) provide all the bits required for reverse and forward resolution to allow name resolution even if the network is disconnected from the Internet. This will also allow operation with DNSSEC if you set up a false trusted server for "." containing only delegations for your forward and reverse zones directly to the top of your administrative control. This should be signed with a key trusted by all of your clients, equivalent to the real key for "`.`".

Return to the BINDv9 Administrator Reference Manual Table of Contents.

# Section 5. BINDv9 Configuration Reference

BINDv9 configuration is broadly similar to BIND 8.x; however, there are a few new areas of configuration, such as views. BIND 8.x configuration files should work with few alterations in BINDv9, 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 4.9.x configuration files can be converted to the new format by using the Perl script **`src/bin/named/named-bootconf.pl`** from the BIND 8 release kit.

## 5.1 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, as described in “Address Match Lists” on page 28. |
| **`domain_name`** | A quoted string which will be used as a DNS name, for example “*my.test.domain*”. |
| **`dotted_decimal`** | One or more integers valued 0 through 255 separated only 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, like **`fe80::200:f8ff:fe01:9742`**. |
| **`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 root-owned processes. 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 “/’’ and then the number of bits in the netmask. E.g. **`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_name`** | A **`domain_name`** representing the name of a shared key, to be used for transaction security. |
| **`number`** | A non-negative integer with an entire range limited by the range of a C language signed integer (2,147,483,647 on a machine with 32 bit integers). Its acceptable value might further be limited by the context in which it is used. |

| | |
|---|---|
| **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**. The maximum value of **size_spec** is that of unsigned long integers on the machine. **unlimited** requests unlimited use, or the maximum available amount. **default** 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. Integer storage overflow is currently silently ignored during conversion of scaled values, resulting in values less than intended, possibly even negative. 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**. |

## 5.1.1 Address Match Lists

### 5.1.1.1 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* } *)*

### 5.1.1.2 Definition and Usage

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

- an IP address (IPv4 or IPv6)
- an IP prefix (in the '/'-notation)
- a key ID, as defined by the key statement
- the name of an address match list previously defined with the **acl** statment
- 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 as a topology, and whether the element was negated.

When used as an access control list, a non-negated match allows access and a negated match denies access. If there is no match, access is denied. The clauses allow-query, allow-transfer, allow-update and blackhole all use address match lists like 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.

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

Because of the first-match aspect of the algorithm, an element that defines a subset of another element in the list should come before the broader element, regardless of whether either is negated. For example, in **`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.

### 5.1.2 Comment Syntax

The BINDv9 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 C, C++, or shell/perl constructs.

#### 5.1.2.1 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

#### 5.1.2.2 Definition and Usage

Comments may appear anywhere that whitespace 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, like C++ comments.

For example:

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

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.

## 5.2 Configuration File Grammar

A BINDv9 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 substatements, 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. See *draft-ietf-dnsind-tsig-13.txt* for more information. |
| **logging** | specifies what the server logs, and where the log messages are sent |
| **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 keys that are preconfigured into the server and implicitly trusted. See RFC 2535 for more information. |
| **view** | defines a view |
| **zone** | defines a zone |

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

### 5.2.1 acl Statement Grammar

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

### 5.2.2 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 IP addresses of all interfaces on the system. |
| **localnets** | Matches any host on a network for which the system has an interface. |

### 5.2.3 control Statement Grammar

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

```
    [unix string permission number owner number group number ;
[unix...;[..]]]
};
```

### 5.2.4 `controls` Statement Definition and Usage

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

A UNIX control channel is a "first in first out" (FIFO) named pipe in the file system, and access to it is controlled by normal file system permissions. It is created by **`named`** with the specified file mode bits (see the **`chmod(1)`** manual page), user and group owner. Note that, unlike **`chmod`**, the mode bits specified for **`permission`** will normally have a leading **`0`** so the number is interpreted as octal. Also note that the user and group ownership specified as owner and group must be given as numbers, not names. It is recommended that the permissions be restricted to administrative personnel only to prevent random users on the system from having the ability to manage the local nameserver.

An **`inet`** control channel is a TCP/IP socket accessible to the Internet, created at the specified **`ip_port`** on the specified **`ip_addr`**. It is recommended that 127.0.0.1 be the only **`ip_addr`** used, and this only if you trust all non-privileged users on the local host to manage your nameserver.

*The **`controls`** statement is not yet implemented in BINDv9. The server always listens for control connections on IP address 127.0.0.1, port 953.*

### 5.2.5 `include` Statement Grammar

```
include "filename";
```

### 5.2.6 `include` Statement Definition and Usage

The **`include`** statement inserts the specified file at the point that 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 a nameserver.

### 5.2.7 `key` Statement Grammar

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

### 5.2.8 **key** Statement Definition and Usage

The **key** statement defines a key ID which can be used in a server statement to associate an authentication method with a particular nameserver.

A key ID must be created with the **key** statement before it can be used in a server definition or an address match list.

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.

The **key** statement is intended for use in transaction security. Unless included in a server statement, it is not used to sign any requests. It is used to verify requests matching the **key_id** and **algorithm_id**, and sign replies to those requests.

### 5.2.9 **logging** statement grammar

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

### 5.2.10 **logging** statement definition and usage

The **logging** statement configures a wide variety of logging options for the nameserver. 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 are multiple **logging** statements in a configuration, the first defined determines the logging, and warnings are issued for the others via the default **syslog**. If there is no **logging** statement, the logging configuration will be:

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

In BINDv9, 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.

#### 5.2.10.1 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 clause that says whether messages selected for the channel go to a file, to a particular syslog facility, or are discarded. It can optionally also limit the message severity level that will be accepted by the channel (default is **info**), and whether to include a **named**-generated time stamp, the category name and/or severity level (default is not to include any).

The word **null** as the destination option for the channel will cause all messages sent to it to be discarded; in that case, other options for the channel are meaningless.

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

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

If you use the **version** 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 3 old versions of the file *lamers.log* then just before it is opened *lamers.log.1* is renamed to *lames.log.2*, *lamers.log.0* is renamed to *lamers.log.1*, and *lamers.log* is renamed to *lamers.log.0*. No rolled versions are kept by default; any existing log file is simply appended. The **unlimited** keyword is synonymous with **99** in current BIND releases.

Example usage of the size and versions options:

```
channel an_example_level {
    file "lamers.log" versions 3 size 20m;
```

```
        print-time yes;
        print-category yes;
    };
```

The argument for the **syslog** clause is a syslog facility as described in the **syslog** manual page. How **syslog** will handle messages sent to this facility is described in the **syslog.conf** manual 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 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 latter method is not yet implemented*). The global debug level can be set to zero, and debugging mode turned off, by running **ndc 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, e.g.

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

will get debugging output of level 3 or less any time the server is in debugging mode, regardless of the global debugging level. Channels with **dynamic** severity use the server's global 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 the section "The category Phrase" on page 36.

```
channel default_syslog {
    syslog daemon;     # send to syslog's daemon facility
    severity info;     # only send priority info and higher
};
channel default_debug {
    file "named.run";  # write to named.run in the working directory
                       # Note: stderr is used instead of "named.run"
                       # if the server is started with the "-f"
                       # option.
    severity dynamic   # log at the server's current debug level
};
channel default_stderr {      # writes to stderr
    file "<stderr>";          # this is illustrative only;
                              # there's currently no way of
                              # specifying an internal file
                              # descriptor in the configuration
                              # language.
    severity info;            # only send priority info and higher
};
channel null {
    null;                     # toss anything sent to this channel
};
```

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

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

Following are the available categories and brief descriptions of the types of log information they contain. *This list is still subject to change.*

| | |
|---|---|
| **default** | The default category defines the logging options for those categories where no specific configuration has been defined. If you do not define a default category, the following definition is used: `category default { default_syslog; default_debug; };` |
| **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. |
| **network** | Network operations. |
| **update** | Dynamic updates. |

### 5.2.11 options Statement Grammar

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

```
options {
    [ version version_string; ]
    [ directory path_name; ]
    [ named-xfer path_name; ]
    [ tkey-domain string; ]
    [ tkey-dhkey string number; ]
    [ dump-file path_name; ]
    [ memstatistics-file path_name; ]
    [ pid-file path_name; ]
    [ statistics-file path_name; ]
    [ auth-nxdomain yes_or_no; ]
    [ deallocate-on-exit yes_or_no; ]
```

```
    [ dialup yes_or_no; ]
    [ fake-iquery yes_or_no; ]
    [ fetch-glue yes_or_no; ]
    [ has-old-clients yes_or_no; ]
    [ host-statistics yes_or_no; ]
    [ multiple-cnames yes_or_no; ]
    [ notify yes_or_no; ]
    [ recursion yes_or_no; ]
    [ rfc2308-type1 yes_or_no; ]
    [ use-id-pool yes_or_no; ]
    [ maintain-ixfr-base yes_or_no; ]
    [ forward ( only | first ); ]
    [ forwarders { [ in_addr ; [ in_addr ; ... ] ] }; ]
   [ check-names ( master | slave | response ) ( warn | fail | ignore);
]
    [ allow-query { address_match_list }; ]
    [ allow-transfer { address_match_list }; ]
    [ allow-recursion { address_match_list }; ]
    [ blackhole { address_match_list }; ]
    [ listen-on [ port ip_port ] { address_match_list }; ]
    [ query-source [ 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-queries number; ]
    [ transfer-format ( one-answer | many-answers ); ]
    [ transfers-in  number; ]
    [ transfers-out number; ]
    [ transfers-per-ns number; ]
    [ transfer-source ip_addr; ]
    [ also-notify { ip_addr; [ ip_addr; ... ] };
    [ max-ixfr-log-size number; ]
    [ coresize size_spec ; ]
    [ datasize size_spec ; ]
    [ files size_spec ; ]
    [ stacksize size_spec ; ]
    [ cleaning-interval number; ]
    [ heartbeat-interval number; ]
    [ interface-interval number; ]
    [ statistics-interval number; ]
    [ topology { address_match_list }; ]
    [ sortlist { address_match_list }; ]
    [ rrset-order { order_spec ; [ order_spec ; ... ] ] };
    [ lame-ttl number; ]
    [ max-ncache-ttl number; ]
    [ min-roots number; ]
```

```
    [ use-ixfr yes_or_no ; ]
    [ treat-cr-as-space yes_or_no ; ]
};
```

### 5.2.12 `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 more than one occurrence is found, the first occurrence determines the actual options used, and a warning will be generated. If there is no **options** statement, an options block with each option set to its default will be used.

| | |
|---|---|
| **version** | The version the server should report via a query of name **version.bind** in class **chaos**. The default is the real version number of this server. |
| **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. |
| **named-xfer** | *This option is obsolete.* It was used in BIND 8 to specify the pathname to the **named-xfer** program. In BINDv9, no separate **named-xfer** program is needed; its functionality is built into the name server. |
| **dump-file** | The pathname of the file the server dumps the database to when it receives **SIGINT** signal (**ndc dumpdb**). If not specified, the default is "**named_dump.db**". *Not yet implemented in BINDv9.* |
| **memstatistics-file** | The pathname of the file the server writes memory usage statistics to on exit. If not specified, the default is "**named.memstats**". *Not yet implemented in BINDv9.* |
| **pid-file** | The pathname of the file the server writes its process ID in. If not specified, the default is operating system dependent, but is usually */var/run/named.pid* or */etc/named.pid*. The pid-file is used by programs that want to send signals to the running nameserver. |
| **statistics-file** | The pathname of the file the server appends statistics to. If not specified, the default is "**named.stats**". *Not yet implemented in BINDv9*. |

#### 5.2.12.1 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. BINDv9 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 **`zone`** statement, in which case it overrides the **`options dialup`** statement.

If the zone is a **`master`** then the server will send out a NOTIFY request to all the slaves. This will 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.

If the zone is a **`slave`** or **`stub`** then the server will suppress the regular "zone up to date" queries and only perform them when the **`heartbeat-interval`** expires. *Not yet implemented in BINDv9.*

**`fake-iquery`** In BIND 8, this option was used to enable simulating the obsolete DNS query type IQUERY. BINDv9 never does IQUERY simulation.

**fetch-glue** — If **yes** (the default), the server will fetch "glue" resource records it doesn't have when constructing the additional data section of a response. (Information present outside of the authoritative nodes in the zone is called "glue" information). **fetch-glue no** can be used in conjunction with **recursion no** to prevent the server's cache from growing or becoming corrupted (at the cost of requiring more work from the client). *Not yet implemented in BINDv9.*

**has-old-clients** — This option was incorrectly implemented in BIND 8, and is ignored by BINDv9. To achieve the intended effect of **has-old-clients yes**, specify the two separate options **auth-nxdomain yes** and **rfc2308-type-1 no** instead.

**host-statistics** — If **yes**, then statistics are kept for every host that the nameserver interacts with. The default is **no**. Note: turning on **host-statistics** can consume huge amounts of memory. *Not yet implemented in BINDv9.*

**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. BINDv9 maintains a transaction log whenever possible. If you need to disable outgoing incremental zone transfers, use **provide-ixfr no**.

**multiple-cnames** — This option was used in BIND 8 to allow a domain name to allow multiple CNAME records in violation of the DNS standards. BINDv9 currently does not check for multiple CNAMEs in zone data loaded from master files, but such checks may be introduced in a later release. BINDv9 always strictly enforces the CNAME rules in dynamic updates.

**notify**
: If **yes** (the default), DNS NOTIFY messages are sent when a zone the server is authoritative for changes. The use of NOTIFY speeds synchronization between the master and its slaves. Slave servers that receive a NOTIFY message and understand it will contact the master server for the zone and see if they need to do a zone transfer, and if they do, they will initiate it immediately. The **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. *Not yet supported in BINDv9.*

**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 not on, the server will return a referral to the client if it doesn't know the answer. The default is **yes**. See also **fetch-glue** above.

**rfc2308-type1**
: If **yes**, the server will send NS records along with the SOA record for negative answers. You need to set this to **no** if you have an old BIND server using you as a forwarder that does not understand negative answers which contain both SOA and NS records or you have an old version of sendmail. The correct fix is to upgrade the broken server or sendmail. The default is **no**. *Not yet implemented in BINDv9.*

**use-id-pool**
: *This option is obsolete.* BINDv9 always allocates query IDs from a pool.

**treat-cr-as-space**
: This option was used in BIND 8 to make the server treat '**\r**' characters the same way as **<space>** " " or '**\t**', to facilitate loading of zone files on a UNIX system that were generated on an NT or DOS machine. In BINDv9, both UNIX '**\n**' and NT/DOS '**\r\n**' newlines are always accepted, and the option is ignored.

#### 5.2.12.2 Forwarding

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

| | |
|---|---|
| **`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 different **`forward only/first behavior`**, or not forward at all. See “zone Statement Grammar” on page 56 for more information.

### 5.2.12.3 Name Checking

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

Three checking methods are available:

| | |
|---|---|
| **`ignore`** | No checking is done. |
| **`warn`** | Names are checked against their expected client contexts. Invalid names are logged, but processing continues normally. |
| **`fail`** | Names are checked against their expected client contexts. Invalid names are logged, and the offending data is rejected. |

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

The defaults are:

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

**`check-names`** may also be specified in the **`zone`** statement, in which case it overrides the **`options check-names`** statement. When used in a **`zone`**

statement, the area is not specified (because it can be deduced from the zone type).

*Name checking is not yet implemented in BINDv9.*

#### 5.2.12.4 Access Control

Access to the server can be restricted based on the IP address of the requesting system. See “Address Match Lists” on page 28 for details on how to specify IP address lists.

| | |
|---|---|
| **allow-query** | Specifies which hosts are allowed to ask ordinary 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. |
| **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 from 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**. *Not yet implemented in BINDv9.* |

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

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

The listen-on option only applies to IPv4. Currently, the server always listens for IPv6 requests on a wildcard address and port 53. A separate **listen-on-v6** option may be added in a later release.

#### 5.2.12.6 Query Address

If the server doesn't know the answer to a question, it will query other nameservers. **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 ***** 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 defaults are

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

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

#### 5.2.12.7 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.

| | |
|---|---|
| **max-transfer-time-in** | Inbound zone transfers running longer than this many minutes will be terminated. The default is 120 minutes (2 hours). |
| **max-transfer-idle-in** | Inbound zone transfers making no progress in this many minutes will be terminated. The default is 60 minutes (1 hour). |
| **max-transfer-time-out** | Outbound zone transfers running longer than this many minutes will be terminated. The default is 120 minutes (2 hours). |
| **max-transfer-idle-out** | Outbound zone transfers making no progress in this many minutes will be terminated. The default is 60 minutes (1 hour). |

**transfer-format**
The server supports two zone transfer methods. **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 BINDv9, BIND 8.x and patched versions of BIND 4.9.5. The default is **one-answer**. **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 nameserver. 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 nameserver. **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. 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-zone basis by including a **`transfer-source`** statement within the **`zone`** block in the configuration file. |
| **`transfer-source-v6`** | Like **`transfer-source`**, but for zone transfers performed using IPv6. |
| **`serial-queries`** | 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, but more importantly each query uses a small amount of memory in the slave server while waiting for the master server to respond. The **`serial-queries`** option sets the maximum number of concurrent serial-number queries allowed to be outstanding at any given time. The default is 4. Note: If a server loads a large (tens or hundreds of thousands) number of slave zones, then this limit should be raised to the high hundreds or low thousands -- otherwise the slave server may never actually become aware of zone changes in the master servers. Beware, though, that setting this limit arbitrarily high can spend a considerable amount of your slave server's network, CPU, and memory resources. As with all tunable limits, this one should be changed gently and monitored for its effects. *Not yet implemented in BINDv9.* |

| | |
|---|---|
| **also-notify** | Defines a global list of IP addresses that are also sent NOTIFY messages whenever a fresh copy of the zone is loaded. This helps to ensure that copies of the zones will quickly converge on "stealth" servers. If an **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). *Not yet implemented in BINDv9.* |

#### 5.2.12.8 Resource Limits

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

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 "Configuration File Grammar" on page 30 for more details.

| | |
|---|---|
| **coresize** | The maximum size of a core dump. The default is **default**. *Not yet implemented in BINDv9.* |
| **datasize** | The maximum amount of data memory the server may use. The default is **default**. *Not yet implemented in BINDv9.* |

| | |
|---|---|
| **`files`** | The maximum number of files the server may have open concurrently. The default is **`unlimited`**. Note: on some operating systems the server cannot set an unlimited value and cannot determine the maximum number of open files the kernel can support. On such systems, choosing **`unlimited`** will cause the server to use the larger of the **`rlim_max`** for **`RLIMIT_NOFILE`** and the value returned by **`sysconf(_SC_OPEN_MAX)`**. If the actual kernel limit is larger than this value, use **`limit files`** to specify the limit explicitly. *Not yet implemented in BINDv9.* |
| **`max-ixfr-log-size`** | The **`max-ixfr-log-size`** will be used in a future release of the server to limit the size of the transaction log kept for Incremental Zone Transfer. *Not yet implemented in BINDv9.* |
| **`stacksize`** | The maximum amount of stack memory the server may use. The default is **`default`**. *Not yet implemented in BINDv9.* |
| **`tcp-clients`** | The maximum number of simultaneous client TCP connections that the server will accept. The default is 100. |
| **`recursive-clients`** | The maximum number of simultaneous recursive lookup the server will perform on behalf of clients. The default is 100. |

*Resource limits are not yet implemented in BINDv9.*

#### 5.2.12.9 Periodic Task Intervals

| | |
|---|---|
| **`cleaning-interval`** | The server will remove expired resource records from the cache every **`cleaning-interval`** minutes. The default is 60 minutes. If set to 0, no periodic cleaning will occur. |
| **`heartbeat-interval`** | The server will perform zone maintenance tasks for all zones marked **`dialup yes`** whenever this interval expires. The default is 60 minutes. Reasonable values are up to 1 day (1440 minutes). If set to 0, no zone maintenance for these zones will occur. *Not yet implemented in BINDv9.* |

| | |
|---|---|
| **`interface-interval`** | The server will scan the network interface list every **`interface-interval`** minutes. The default is 60 minutes. If set to 0, interface scanning will only occur when the configuration file is loaded. After the scan, listeners will be started on any new interfaces (provided they are allowed by the **`listen-on`** configuration). Listeners on interfaces that have gone away will be cleaned up. |
| **`statistics-interval`** | Nameserver statistics will be logged every **`statistics-interval`** minutes. The default is 60. If set to 0, no statistics will be logged. *Not yet implemented in BINDv9.* |

#### 5.2.12.10 Topology

All other things being equal, when the server chooses a nameserver to query from a list of nameservers, it prefers the one that is topologically closest to itself. The **`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; };
```

*The **`topology`** option is not yet implemented in BINDv9.*

#### 5.2.12.11 The `sortlist` Statement

Resource Records (RRs) are the data associated with the names in a domain name space. The data is maintained in the form of sets of RRs. The order of RRs in a set is, by default, not significant. Therefore, to control the sorting of records in a set resource records, or *RRset*, you must use the **`sortlist`** statement.

RRs are explained more fully in See "Types of Resource Records and When to Use Them" on page 61.. Specifications for RRs are documented in RFC 1035.

When returning multiple RRs, the nameserver will normally return them in *Round Robin* order, i.e. after each request, the first RR is put at the end of the list. The client resolver code should rearrange the RRs as appropriate, i.e. using any addresses on the local net in preference to other addresses. However, not all resolvers can do this or are correctly configured. When a client is using a local server the sorting can be performed in the server, based on the client's address. This only requires configuring the nameservers, not all the clients.

The **sortlist** statement (see below) takes an **address_match_list** and interprets it even more specifically than the **topology** statement does (see "Topology" on page 50). 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 like the **address_match_list** in a **topology** statement. Each top level element is assigned a distance and the address in the response with the minimum distance is moved to the beginning of the response.

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

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

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

The following example will give reasonable behavior for the local host and hosts on directly connected networks. It is similar to the behavior of the address sort in BIND 8.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; };
};
```

*The **`sortlist`** option is not yet implemented in BINDv9.*

#### 5.2.12.12 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. For example, the records for a zone might be configured always to be returned in the order they are defined in the zone file. Or perhaps a random shuffle of the records as they are returned is wanted. The **`rrset-order`** statement permits configuration of the ordering made of the records in a multiple record response. The default, if no ordering is defined, is a cyclic ordering (round robin).

An **`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 "**`*`**".

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.

If no **rrset-order** statement is specified, then a default one of:

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

is used.

*The **rrset-order** statement is not yet implemented in BINDv9.*

#### 5.2.12.13 Tuning

| | |
|---|---|
| **lame-ttl** | Sets the number of seconds to cache a lame server indication. 0 disables caching. (This is NOT recommended.) Default is 600 (10 minutes). Maximum value is 1800 (30 minutes). *Not yet implemented in BINDv9.* |
| **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 the maximum retention time for ordinary (positive) answers (7 days) and will be silently truncated to 7 days if set to a value which is greater that 7 days. *Not yet implemented in BINDv9.* |
| **min-roots** | The minimum number of root servers that is required for a request for the root servers to be accepted. Default is 2. *Not yet implemented in BINDv9.* |

#### 5.2.12.14 Deprecated Features

**use-ixfr** is deprecated in BINDv9. If you need to disable IXFR to a particular server or servers see information on the **provide-ixfr** option in the Server Statement description ("server Statement Grammar" on page 54 , below) and in the description of Incremental Transfer (IXFR) (“Incremental Transfer (IXFR)” on page 13).

### 5.2.13 `server` Statement Grammar

```
server ip_addr {
    [ bogus yes_or_no ; ]
    [ provide-ixfr yes_or_no ; ]
    [ request-ixfr yes_or_no ; ]
    [ transfers number ; ]
    [ transfer-format (one-answer | many-answers) ; ]
    [ keys { string ; [ string ; [...]] } ; ]
};
```

### 5.2.14 `server` Statement Definition and Usage

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

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 **bogus** clause is not yet implemented in BINDv9.*

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 nonincremental. If not set, the value of the **provide-ixfr** option in the 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 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 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 in-bound zone transfers from the specified server.

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

### 5.2.15 trusted-keys Statement Grammar

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

### 5.2.16 trusted-keys Statement Definition and Usage

The trusted-keys statement is for use with DNSSEC-style security, originally specified in RFC 2065. DNSSEC is meant to provide three distinct services: key distribution, data origin authentication, and transaction and request authentication. A complete description of DNSSEC and its use is beyond the scope of this document, and readers interested in more information should start with RFC 2065 and then continue with the relevant *Internet Drafts* (IDs) documents. A list of the Internet Drafts pertaining to DNSSEC can be found in “Internet Drafts” on page 79 in Appendix C of this document. (Their filenames begin with “draft-ietf-dnssec.”). IDs are RFCs in their preliminary stages of development—they are the working drafts of IETF working groups—and can be obtained via anonymous **FTP** from *ftp://www.isi.edu/internet-drafts/ or ftp://www.ietf.org/rfcs/*.

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

A trusted key is added when a public key for a non-authoritative zone is known, but cannot be securely obtained through DNS. This occurs when a signed zone is a child of an unsigned zone. Adding the trusted key here allows data signed by that zone to be considered secure.

### 5.2.17 view Statement Grammar

```
view “name” {
      . . .
     [zone_statement; [zone_statement; [....]]
};
```

### 5.2.18 `view` Statement Definition and Usage

**view** statements are used to provide a different view of the same namespace to different clients. *They are not yet fully implemented.*

### 5.2.19 `zone` Statement Grammar

```
zone string [class] [{
    type (master|slave|hint|stub|forward) ;
    [ allow-query { address_match_list } ; ]
    [ allow-transfer { address_match_list } ; ]
    [ allow-update { address_match_list } ; ]
    [ update-policy { update_policy_rule [...] } ; ]
    [ allow-update-forwarding { address_match_list } ; ]
    [ also-notify { [ ip_addr ; [ip_addr ; [...]]] } ; ]
    [ check-names (warn|fail|ignore) ; ]
    [ dialup true_or_false ; ]
    [ file string ; ]
    [ forward (only|first) ; ]
    [ forwarders { [ ip_addr ; [ ip_addr ; [...]]] } ; ]
    [ ixfr-base string ; ]
    [ ixfr-tmp-file string ; ]
    [ maintain-ixfr-base true_or_false ; ]
    [ masters [port number] { ip_addr ; [ip_addr ; [...]] } ; ]
    [ 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 true_or_false ; ]
    [ pubkey number number number string ; ]
    [ transfer-source (ip_addr | *) ; ]
}];
```

## 5.2.20 `zone` Statement Definition and Usage

### 5.2.20.1 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 that the slave contacts to update its copy of the zone. If a port is specified, the slave then checks to see if the zone is current and zone transfers will be done to the port given. 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 start-up 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 100K files into a single directory.)

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

**`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, cancelling 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 (i.e., "forward first to, " then "forward only," or vice versa, but want to use the same servers as set globally) you need to respecify the global forwarders.

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

### 5.2.20.2 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 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 was CHAOSnet, a LAN protocol created in the mid-1970s. Zone data for it can be specified with the **chaos** class.

#### 5.2.20.3 Zone Options

| | |
|---|---|
| **allow-query** | See the description of **allow-query** under "Access Control" on page 44. |
| **allow-transfer** | See the description of **allow-transfer** under "Access Control" on page 44. |
| **allow-update** | Specifies which hosts are allowed to submit Dynamic DNS updates for master zones. The default is to deny updates from all hosts. |
| **update-policy** | Specifies a "Simple Secure Update" policy. See description below. |
| **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 to deny update forwarding from all hosts. *Update forwarding is not yet implemented.* |
| **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 nameservers for the zone (other than the primary master) plus any IP addresses specified with **also-notify**.<br>**also-notify** is not meaningful for stub zones. The default is the empty list.<br>*Not yet implemented in BINDv9.* |
| **check-names** | See "Name Checking" on page 43.<br>*Not yet implemented in BINDv9.* |
| **dialup** | See the description of **dialup** under "Boolean Options" on page 40.<br>*Not yet implemented in BINDv9.* |
| **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.<br>*Not yet implemented in BINDv9.* |
| **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; the global options are not used.<br>*Not yet implemented in BINDv9.* |
| **ixfr-base** | Specifies the file name for the transaction log file used for dynamic update and IXFR. |
| **max-transfer-time-in** | See the description of **max-transfer-time-in** under "Zone Transfers" on page 45. |

| | |
|---|---|
| **max-transfer-idle-in** | See the description of **max-transfer-idle-in** under "Zone Transfers" on page 45. |
| **max-transfer-time-out** | See the description of **max-transfer-time-outn** under "Zone Transfers" on page 45. |
| **max-transfer-idle-out** | See the description of **max-transfer-idle-out** under "Zone Transfers" on page 45. |
| **notify** | See the description of **notify** under "Boolean Options" on page 40. |
| **pubkey** | Represents a public key for this zone. It is needed when this is the top level authoritative zone served by this server and there is no chain of trust to a trusted key. It is considered secure, so that data that it signs will be considered secure. The DNSSEC flags, protocol, and algorithm are specified, as well as a base-64 encoded string representing the key. |
| **transfer-source** | Determines which local address will be bound to the TCP connection used to fetch this zone. If not set, it defaults to a system controlled value which will usually be the address of the interface *closest to* the remote end. This address must appear in the remote end's **allow-transfer** option for this zone if one is specified. |

#### 5.2.20.4 Dynamic Update Policies

BINDv9 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 BINDv9 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 meaninful 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.

A rule defition looks like:

( **`grant`** | **`deny`** ) *identity nametype name [ types ]*

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

The identity field specifies a name or a wildcard name. The nametype field has 4 values: *name*, *subdomain*, *wildcard*, and *self*..

| | |
|---|---|
| *name* | Matches when the updated name is the same as the name in the name field. |
| *subdomain* | Matches when the updated name is a subdomain of the name in the name field. |
| *wildcard* | Matches when the updated name is a valid expansion of the wildcard name in the name field. |
| *self* | Matches when the updated name is the same as the message signer. The name field is ignored. |

If no types are specified, the rule matches all types except SIG, NS, SOA, and NXT. Types may be specified by name, including "any" (which matches all types except NXT, which can never be updated).

## 5.3 Zone File

### 5.3.1 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.

#### 5.3.1.1 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 nameservers, 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 “The sortlist Statement” on page 50 and “RRset Ordering” on page 52 for details.

The components of a RR are

| | |
|---|---|
| owner name | the domain name where the RR is found. |
| type | an encoded 16 bit value that specifies the type of the resource in this resource record. Types refer to abstract resources. |
| 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 type and sometimes class-dependent data that describes the resource. |

The following are *types* of valid RRs (some of these listed, although not obsolete, are experimental (x) or historical (h) and no longer in general use):

| | |
|---|---|
| A | a host address. |
| A6 | an IPv6 address. |
| AAAA | Obsolete format of IPv6 address |
| AFSDB | (x) location of AFS database servers. Experimental. |
| CNAME | identifies the canonical name of an alias. |
| DNAME | for delegation of reverse addresses. Replaces the domain name specified with another name to be looked up. Described in RFC 2672. |
| HINFO | identifies the CPU and OS used by a host. |
| ISDN | (x) representation of ISDN addresses. Experimental. |
| KEY | stores a public key associated with a DNS name. |
| LOC | (x) for storing GPS info. See RFC 1876. Experimental. |
| MX | identifies a mail exchange for the domain. See RFC 974 for details. |
| NS | the authoritative nameserver for the domain. |
| 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. See RFC 2535 for details. |
| PTR | a pointer to another part of the domain name space. |
| RP | (x) information on persons responsible for the domain. Experimental. |
| RT | (x) route-through binding for hosts that do not have their own direct wide area network addresses. Experimental. |

| | |
|---|---|
| SIG | (“signature”) contains data authenticated in the secure DNS. See RFC 2535 for details. |
| SOA | identifies the start of a zone of authority. |
| SRV | information about well known network services (replaces WKS). |
| WKS | (h) information about which well known network services, such as SMTP, that a domain supports. Historical, replaced by newer RR SRV. |
| X25 | (x) representation of X.25 network addresses. Experimental. |

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

| | |
|---|---|
| IN | the Internet system. |

For information about other, older classes of RRs, See Appendix B, “Historical DNS Information,” on page 75.

*RDATA* is the type-dependent or class-dependent data that describes the resource:

| | |
|---|---|
| A | for the IN class, a 32 bit IP address |
| A6 | maps a domain name to an IPv6 address, with a provision for indirection for leading “prefix” bits. |
| CNAME | a domain name |
| DNAME | provides alternate naming to an entire subtree of the domain name space, rather than to a single node. It causes some suffix of a queried name to be substituted with a name from the DNAME record’s RDATA. |
| MX | a 16 bit preference value (lower is better) followed by a host name willing to act as a mail exchange for the owner domain. |
| NS | a fully qualified domain name. |
| PTR | a fully qualified doman name. |
| SOA | several fields. |

The owner name is often implicit, rather than forming an integral part of the RR. For example, many nameservers 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.

#### 5.3.1.2 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 nameserver 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.

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

Similarly we might see:

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

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

### 5.3.2 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 datum, a domain name matched with relevant data, 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—a 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.

For example:

```
example.com.        IN   MX   10         mail.foo.com.
                    IN   MX   10         mail2.foo.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
```

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

### 5.3.3 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.<br>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, e.g. *1h30m*.

### 5.3.4 Inverse Mapping in IPv4

Reverse name resolution (i.e., 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.)

### 5.3.5 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.**

#### 5.3.5.1 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.
```

#### 5.3.5.2 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.

*NOTE*: The behavior when **origin** is specified differs from that described in RFC 1035. The origin and current domain revert to the values they were prior to the **$INCLUDE** once the file has been read.

#### 5.3.5.3 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.

### 5.3.6 BIND Master File Extension: the $GENERATE Directive

**$GENERATE**

Syntax: **$GENERATE <range> <lhs> <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 **`$`** symbols within the **`lhs`** side are replaced by the iterator value. To get a $ in the output use a double **`$`**, e.g. **`$$`**. If the **`lhs`** is not absolute, the current **`$ORIGIN`** is appended to the name. |
| **`type`** | At present the only supported types are PTR, CNAME and NS. |
| **`rhs`** | rhs is 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. *It is not yet implemented in BINDv9.*

### 5.3.7 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. |
| **`SIGQUIT`** | Causes the server to clean up and exit. |

Return to the BINDv9 Administrator Reference Manual Table of Contents.

# Section 6. Security Considerations

## 6.1 Access Control Lists

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

Using ACLs allows you to have finer control over who can access your nameserver, 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 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; 172.16.0.0/12; 192.168.0.0/16; };
```

// 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*

## 6.2 `chroot` and `setuid` (for UNIX servers)

On UNIX servers, it is possible to run BIND in a *chrooted* environment (**chroot()**) 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 a nonprivileged user ( **`-u`** <user> ). We suggest running as a nonprivileged 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
```

### 6.2.1 The `chroot` environment

In order for a **`chroot()`** environment to work properly in a particular directory (e.g. */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 */dev/null*, and any library directories and files that BIND needs to run on your system. Please consult your operating system's instructions if you need help figuring out which library files you need to copy over to the **`chroot()`** sandbox.

If you are running an operating system that supports static binaries, you can also compile BIND staticly and avoid the need to copy system libraries over to your **`chroot()`** sandbox.

### 6.2.2 Using `setuid`

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.

## 6.3 Dynamic updates

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. BINDv9 also supports authenticating updates cryptographically by means of transaction signatures (TSIG). The use of TSIG is strongly recommended.

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.

# Section 7. Troubleshooting

## 7.1 Common Log Messages and What They Mean

- lame server

```
ns named[111]: Lame server on 'www.foo.com' (in 'foo.com'?): [192.168.0.2].53
'ns2.foo.com'
```

This is a harmless error message. It means that the server at 192.168.0.2 (ns2.foo.com) is listed as a nameserver for “foo.com”, but it doesn’t really know anything about foo.com.

If this is a zone under your control, check each of the nameservers to ensure that they are configured to answer questions properly.

If it’s a zone out on the Internet, it would be nice to notify the owners of the domain in question so that they can take a look at it. In practice, though, not many people have time to do this.

- bad referral

```
ns named[111]: bad referral (other.com !< subdomain.other.com)
```

This indicates that your nameserver (ns.foo.com) queried the nameserver for foo2.com to find out how to get to subdomain.foo2.com. foo2.com told your nameserver that subdomain.foo2.com was delegated to some other.foo2.com, so your nameserver queried that.

someother.foo2.com didn’t think that subdomain.foo2.com had been delegated to it, so it referred your server (ns.foo.com) back to the foo2.com nameserver.

- not authoritative for

```
ns named-xfer[111]: [192.168.0.1] not authoritative for foo.com, SOA query got
rcode 0, aa 0, ancount 1, aucount 0
```

This error usually shows up on a slave server. It indicates that the master server is not answering authoritatively for the zone. This usually happens when the zone is rejected (while named is loading) on the master server. Check the logs on the master server. If ancount -- 0, you may be pointing at the wrong master server for the zone.

- rejected zone

```
ns named[111]: master zone “foo.com” (IN) rejected due to errors (serial111)
```

This indicates that the foo.com zone was rejected because of an error in the zone file. Check the lines above this error -- named will usually tell you what it didn’t like and where to find it in the zone file.

- no NS RRs found

```
ns named[111]: Zone “foo.com” (file foo.com.db): no NS RRs found at zonetop
```

The foo.com.db file is missing NS records at the top of the zone (in the SOA section). Check to make sure they exist and that there is white space (spaces or tabs) in front of them. White spaces matter here.

- no default TTL set

**`ns named[111]: Zone "foo.com" (file foo.com.db): No default TTL set using SOA minimum instead`**

You need to add a $TTL to the top of the foo.com.db zone file. See RFC2308, or section 3.2.3, "Setting TTLs" in this document, for information on how to use $TTL.

- no root nameserver for class

**`findns: No root nameservers for class IN?`**

Your nameserver is having problems finding the root nameservers. Check your root hints file to make sure it is not corrupted. Also, make sure that your nameserver can reach the Internet.

If you are running an internal root nameserver, make sure it's configured properly and is answering queries.

- address already in use

**`ctl_server: bind: Address already in use`**

This usually indicates that another copy of BIND is already running. Verify that you have killed old copies of the daemon.

This can also pop up if you originally ran named as "root" and now run it as a regular user. named may have left behind an ndc control socket that is owned by root if it crashed, or was not killed gracefully.

This means that the regular user wouldn't be able to delete it, so it would think named is still running. The solution is to remove any ndc sockets in /usr/local/etc, or /var/run, etc.

## 7.2 Common Problems

### 7.2.1 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 (see the sample configurations in "Sample Configuration and Logging" on page 9). 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.

## 7.3 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 secondary server is lower than the serial number on the primary, the secondary server will attempt to update its copy of the zone.

Setting the serial number to a lower number on the primary server than the secondary server means that the secondary 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 secondaries have updated to the new zone serial number, then reset the number to what you want it to be, and reload the zone again.

## 7.4 Where Can I Get Help?

The Internet Software Consortium (ISC) offers a wide range of support and service agreements for BIND, DHCP and INN 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, DHCP or INN.

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

# Appendices

# Appendix A. Acknowledgements

## A.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 an 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. Paul 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.

# Appendix B. Historical DNS Information

## B.1 Classes of resource records

### B.1.1 HS = hesiod

### B.1.2 CH = chaos

# Appendix C. Bibliography (and Suggested Reading)

## C.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 themselvers 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/RFCxxx.txt* (where *xxx* is the number of the RFC). RFCs are also available via the Web at *http://www.ietf.org/rfc/*.

### C.1.1 Standards

RFC974. Partridge, C. *Mail Routing and the Domain System*. January 1986. (Standard

RFC1034. Mockapetris, P.V. *Domain Names - Concepts and Facilities*. P.V. November 1987.

RFC1035. Mockapetris, P. V. *Domain Names - Implementation and Specification*. November 1987.

### C.1.2 Proposed Standards

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

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

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

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

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

### C.1.3 Proposed Standards Still Under Development

*Note:* the following list of RFCs are undergoing major revision by the IETF. (See below, *Internet Drafts*, for current versions).

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

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

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

### C.1.4 Other Important RFCs About DNS Implementation

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

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

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

### C.1.5 Resource Record Types

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

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

RFC2168. Danie1,R., M. Mealling. *Resolution of Uniform Resource Identifiers using the Domain Name System. June 1997.*

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

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

RFC2163. Allocchio, A. U*sing the Internet DNS to Distribute MIXER Conformant Global Address Mapping*. January 1998.

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

### C.1.6 DNS and the Internet

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

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

RFC1591. Postel, J. D*omain Name System Structure and Delegation*. March 1994.

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

### C.1.7 DNS Operations

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

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

RFC2182. Elz, R. R. Bush, S. Bradner, M. Patton. *Selection and Operation of Secondary DNS Servers*. July 1997.

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

### C.1.8 Other DNS-related RFCs

*Note: the following list of RFCs, although DNS-related, are not concerned with implementing software.*

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

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

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

RFC2240. Vaughan, O. *A Legal Basis for Domain Name Allocation*. November1997.

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

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

### C.1.9 Obsolete and Unimplemented Experimental RRs

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

## C.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.

IDs can be obtained via **FTP** from *ftp://www.isi.edu/internet-drafts/* or from *http://www.ietf.org/1id-abstracts.html*.

*draft-duerst-dns-i18n-01.txt*
*draft-ietf-dhc-dhcp-dns-10.txt*
*draft-ietf-dnsind-apl-rr-03.txt*
*draft-ietf-dnsind-dddd-01.txt*
*draft-ietf-dnsind-dhcp-rr-00.txt*
*draft-ietf-dnsind-edns1-03.txt*
*draft-ietf-dnsind-iana-dns-04.txt*
*draft-ietf-dnsind-indirect-key-00.txt*
*draft-ietf-dnsind-keyreferral-00.txt*
*draft-ietf-dnsind-kitchen-sink-02.txt*
*draft-ietf-dnsind-local-compression-05.txt*
*draft-ietf-dnsind-local-names-07.txt*
*draft-ietf-dnsind-rfc2052bis-05.txt*
*draft-ietf-dnsind-rollover-00.txt*
*draft-ietf-dnsind-sec-rr-00.txt*
*draft-ietf-dnsind-sigalgopt-00.txt*
*draft-ietf-dnsind-simple-secure-update-02.txt*
*draft-ietf-dnsind-test-tlds-13.txt*
*draft-ietf-dnsind-tkey-01.txt*
*draft-ietf-dnsind-tsig-13.txt*
*draft-ietf-dnsind-verify-00.txt*
*draft-ietf-dnssec-ar-00.txt*
*draft-ietf-dnssec-as-map-05.txt*
*draft-ietf-dnssec-key-handling-00.txt*
*draft-ietf-dnssec-secfail-00.txt*
*draft-ietf-dnssec-update2-00.txt*
*draft-ietf-ipngwg-2292bis-00.txt*
*draft-ietf-ipngwg-dns-lookups-05.txt*
*draft-dunlap-dns-duxfr-00.txt*
*draft-schroeppel-dnsind-ecc-00.txt*
*draft-skwan-gss-tsig-04.txt*
*draft-skwan-utf8-dns-02.txt*

## C.3 Electronic Mail Communication

Wellington, Brian (bwellington@tislabs.com). *DNSSEC usage document*. E-mail to David Conrad (David_Conrad@isc.org). 15 March 1999.

Wellington, Brian (bwellington@tislabs.com). *TSIG guide for BIND 8.2+*. E-mail to private mailing list (private communication). 22 April 1999.

## C.4 Other BIND Documents

Albitz, Paul and Cricket Liu. 1998. *DNS and BIND*. Sebastopol, CA: O'Reilly and Associates.