BIND 9 Administrator Reference Manual
## Contents

1 Introduction
   1.1 Scope of Document ............................................. 1
   1.2 Organization of This Document .................................. 1
   1.3 Conventions Used in This Document .............................. 1
   1.4 The Domain Name System (DNS)
      1.4.1 DNS Fundamentals ......................................... 2
      1.4.2 Domains and Domain Names ................................. 2
      1.4.3 Zones ....................................................... 2
      1.4.4 Authoritative Name Servers ............................... 3
         1.4.4.1 The Primary Master ................................. 3
         1.4.4.2 Slave Servers ........................................ 3
         1.4.4.3 Stealth Servers ...................................... 3
      1.4.5 Caching Name Servers ...................................... 4
         1.4.5.1 Forwarding ............................................ 4
      1.4.6 Name Servers in Multiple Roles ........................... 4

2 BIND Resource Requirements
   2.1 Hardware requirements ......................................... 5
   2.2 CPU Requirements .............................................. 5
   2.3 Memory Requirements .......................................... 5
   2.4 Name Server Intensive Environment Issues .................... 5
   2.5 Supported Operating Systems .................................. 6

3 Name Server Configuration
   3.1 Sample Configurations ......................................... 7
      3.1.1 A Caching-only Name Server ............................... 7
      3.1.2 An Authoritative-only Name Server ....................... 7
   3.2 Load Balancing ................................................. 8
   3.3 Name Server Operations ....................................... 8
      3.3.1 Tools for Use With the Name Server Daemon .............. 8
         3.3.1.1 Diagnostic Tools ................................... 9
         3.3.1.2 Administrative Tools ............................... 9
      3.3.2 Signals .................................................... 11

4 Advanced DNS Features
   4.1 Notify ......................................................... 13
   4.2 Dynamic Update ................................................ 13
      4.2.1 The journal file ......................................... 13
   4.3 Incremental Zone Transfers (IXFR) ............................ 14
   4.4 Split DNS ...................................................... 14
      4.4.1 Example split DNS setup ................................ 15
   4.5 TSIG .......................................................... 17
      4.5.1 Generate Shared Keys for Each Pair of Hosts ........... 17
         4.5.1.1 Automatic Generation ............................... 18
         4.5.1.2 Manual Generation ................................ 18
      4.5.2 Copying the Shared Secret to Both Machines ............ 18
      4.5.3 Informing the Servers of the Key’s Existence ........... 18
      4.5.4 Instructing the Server to Use the Key .................. 18
      4.5.5 TSIG Key Based Access Control ........................... 19
      4.5.6 Errors .................................................... 19
   4.6 TKEY .......................................................... 19
   4.7 SIG(0) ......................................................... 20
   4.8 DNSSEC ......................................................... 20
4.8.1 Generating Keys .................................................. 20
4.8.2 Signing the Zone .................................................. 21
4.8.3 Configuring Servers .............................................. 21
4.9 IPv6 Support in BIND 9 .............................................. 22
4.9.1 Address Lookups Using AAAA Records ......................... 23
4.9.2 Address to Name Lookups Using Nibble Format ................ 23

5 The BIND 9 Lightweight Resolver .............................. 25
5.1 The Lightweight Resolver Library ............................. 25
5.2 Running a Resolver Daemon ..................................... 25

6 BIND 9 Configuration Reference ............................... 27
6.1 Configuration File Elements ..................................... 27
   6.1.1 Address Match Lists ....................................... 29
       6.1.1.1 Syntax .................................................. 29
       6.1.1.2 Definition and Usage ................................. 29
   6.1.2 Comment Syntax ............................................ 29
       6.1.2.1 Syntax .................................................. 30
       6.1.2.2 Definition and Usage ................................ 30
   6.2 Configuration File Grammar .................................. 30
      6.2.1 acl Statement Grammar .................................. 31
      6.2.2 acl Statement Definition and Usage ....................... 31
      6.2.3 controls Statement Grammar ............................ 32
      6.2.4 controls Statement Definition and Usage ................ 32
      6.2.5 include Statement Grammar .............................. 33
      6.2.6 include Statement Definition and Usage ................ 33
      6.2.7 key Statement Grammar .................................. 33
      6.2.8 key Statement Definition and Usage ....................... 33
      6.2.9 logging Statement Grammar ................................ 33
      6.2.10 logging Statement Definition and Usage .................. 34
           6.2.10.1 The channel Phrase ............................... 34
           6.2.10.2 The category Phrase .............................. 36
           6.2.10.3 The query-errors Category ......................... 38
      6.2.11 lwres Statement Grammar ................................ 39
      6.2.12 lwres Statement Definition and Usage .................... 39
      6.2.13 masters Statement Grammar ................................ 39
      6.2.14 masters Statement Definition and Usage .................. 39
      6.2.15 options Statement Grammar ................................ 40
      6.2.16 options Statement Definition and Usage .................. 42
           6.2.16.1 Boolean Options .................................... 45
           6.2.16.2 Forwarding ......................................... 49
           6.2.16.3 Dual-stack Servers ................................ 50
           6.2.16.4 Access Control ..................................... 50
           6.2.16.5 Interfaces .......................................... 51
           6.2.16.6 Query Address ..................................... 52
           6.2.16.7 Zone Transfers ..................................... 54
           6.2.16.8 UDP Port Lists ..................................... 56
           6.2.16.9 Operating System Resource Limits ................. 56
           6.2.16.10 Server Resource Limits ............................ 57
           6.2.16.11 Periodic Task Intervals ........................... 57
           6.2.16.12 Topology .......................................... 58
           6.2.16.13 The sortlist Statement ............................ 59
           6.2.16.14 RSet Ordering ..................................... 60
           6.2.16.15 Tuning ............................................. 60
           6.2.16.16 Built-in server information zones .................. 62
           6.2.16.17 Built-in Empty Zones .............................. 63
           6.2.16.18 Additional Section Caching ......................... 66
      6.2.17 server Statement Grammar ................................ 67
      6.2.18 server Statement Definition and Usage .................... 67
| B.1  | dig                          | 103 |
| B.2  | host                         | 108 |
| B.3  | dnssec-dsfromkey             | 110 |
| B.4  | dnssec-keyfromlabel          | 111 |
| B.5  | dnssec-keygen                | 113 |
| B.6  | dnssec-signzone              | 115 |
| B.7  | named-checkconf              | 119 |
| B.8  | named-checkzone              | 120 |
| B.9  | named                        | 122 |
| B.10 | nsupdate                     | 125 |
| B.11 | rndc                         | 129 |
| B.12 | rndc.conf                    | 132 |
| B.13 | rndc-confgen                 | 134 |
Chapter 1

Introduction

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

1.1 Scope of Document

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

This version of the manual corresponds to BIND version 9.6.

1.2 Organization of This Document

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

1.3 Conventions Used in This Document

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

<table>
<thead>
<tr>
<th>To describe:</th>
<th>We use the style:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a pathname, filename, URL, hostname, mailing list name, or new term or concept</td>
<td>Fixed width</td>
</tr>
<tr>
<td>literal user input</td>
<td><strong>Fixed Width Bold</strong></td>
</tr>
<tr>
<td>program output</td>
<td>Fixed Width</td>
</tr>
</tbody>
</table>
The purpose of this document is to explain the installation and upkeep of the BIND (Berkeley Internet Name Domain) software package, and we begin by reviewing the fundamentals of the Domain Name System (DNS) as they relate to BIND.

1.4.1 DNS Fundamentals

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

Clients look up information in the DNS by calling a resolver library, which sends queries to one or more name servers and interprets the responses. The BIND 9 software distribution contains a name server, named, and a resolver library, liblwres. The older libbind resolver library is also available from ISC as a separate download.

1.4.2 Domains and Domain Names

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

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

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

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

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

1.4.3 Zones

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

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

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

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

### 1.4.4 Authoritative Name Servers

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

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

#### 1.4.4.1 The Primary Master

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

In some cases, however, the master file may not be edited by humans at all, but may instead be the result of dynamic update operations.

#### 1.4.4.2 Slave Servers

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

#### 1.4.4.3 Stealth Servers

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

A stealth server is a server that is authoritative for a zone but is not listed in that zone’s NS records. Stealth servers can be used for keeping a local copy of a zone to speed up access to the zone’s records or to make sure that the zone is available even if all the “official” servers for the zone are inaccessible.
A configuration where the primary master server itself is a stealth server is often referred to as a "hidden primary" configuration. One use for this configuration is when the primary master is behind a firewall and therefore unable to communicate directly with the outside world.

### 1.4.5 Caching Name Servers

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

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

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

#### 1.4.5.1 Forwarding

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

There may be one or more forwarders, and they are queried in turn until the list is exhausted or an answer is found. Forwarders are typically used when you do not wish all the servers at a given site to interact directly with the rest of the Internet servers. A typical scenario would involve a number of internal DNS servers and an Internet firewall. Servers unable to pass packets through the firewall would forward to the server that can do it, and that server would query the Internet DNS servers on the internal server’s behalf.

### 1.4.6 Name Servers in Multiple Roles

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

However, since the functions of authoritative name service and caching/recursive name service are logically separate, it is often advantageous to run them on separate server machines. A server that only provides authoritative name service (an *authoritative-only* server) can run with recursion disabled, improving reliability and security. A server that is not authoritative for any zones and only provides recursive service to local clients (a *caching-only* server) does not need to be reachable from the Internet at large and can be placed inside a firewall.
Chapter 2

BIND Resource Requirements

2.1 Hardware requirements

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

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

2.2 CPU Requirements

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

2.3 Memory Requirements

The memory of the server has to be large enough to fit the cache and zones loaded off disk. The max-cache-size option can be used to limit the amount of memory used by the cache, at the expense of reducing cache hit rates and causing more DNS traffic. Additionally, if additional section caching (Section 6.2.16.18) is enabled, the max-acache-size option can be used to limit the amount of memory used by the mechanism. It is still good practice to have enough memory to load all zone and cache data into memory — unfortunately, the best way to determine this for a given installation is to watch the name server in operation. After a few weeks the server process should reach a relatively stable size where entries are expiring from the cache as fast as they are being inserted.

2.4 Name Server Intensive Environment Issues

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

ISC BIND 9 compiles and runs on a large number of Unix-like operating systems and on NT-derived versions of Microsoft Windows such as Windows 2000 and Windows XP. For an up-to-date list of supported systems, see the README file in the top level directory of the BIND 9 source distribution.
Chapter 3

Name Server Configuration

In this chapter we provide some suggested configurations along with guidelines for their use. We suggest reasonable values for certain option settings.

3.1 Sample Configurations

3.1.1 A Caching-only Name Server

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

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

3.1.2 An Authoritative-only Name Server

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

```plaintext
options {
    directory "/etc/namedb";  // Working directory
    allow-query-cache { none; };  // Do not allow access to cache
    allow-query { any; };  // This is the default
    recursion no;  // Do not provide recursive service
}
```

// Provide a reverse mapping for the loopback address 127.0.0.1
zone "0.0.127.in-addr.arpa" {
    type master;
    file "localhost.rev";
    notify no;
};
// We are the master server for example.com
zone "example.com" {
    type master;
    file "example.com.db";
    // IP addresses of slave servers allowed to transfer example.com
    allow-transfer {
        192.168.4.14;
        192.168.5.53;
    };
};
// We are a slave server for eng.example.com
zone "eng.example.com" {
    type slave;
    file "eng.example.com.bk";
    // IP address of eng.example.com master server
    masters { 192.168.4.12; }
};

3.2 Load Balancing

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>TTL</th>
<th>CLASS</th>
<th>TYPE</th>
<th>Resource Record (RR) Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>www</td>
<td>600</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.2</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>IN</td>
<td>A</td>
<td>10.0.0.3</td>
</tr>
</tbody>
</table>

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

For more detail on ordering responses, check the `rrset-order` sub-statement in the `options` statement, see RRset Ordering.

3.3 Name Server Operations

3.3.1 Tools for Use With the Name Server Daemon

This section describes several indispensable diagnostic, administrative and monitoring tools available to the system administrator for controlling and debugging the name server daemon.
3.3.1 Diagnostic Tools

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

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

**Usage**

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

The usual simple use of `dig` will take the form

```
dig @server domain query-type query-class
```

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

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

**Usage**

```
```

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

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

**Usage**

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

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

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

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

3.3.1.2 Administrative Tools

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

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

**Usage**

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

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

**Usage**


named-checkzone  Similar to named-checkzone, but it always dumps the zone content to a specified file (typically in a different format).

rndc  The remote name daemon control (rndc) program allows the system administrator to control the operation of a name server. Since BIND 9.2, rndc supports all the commands of the BIND 8 ndc utility except ndc start and ndc restart, which were also not supported in ndc’s channel mode. If you run rndc without any options it will display a usage message as follows:

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

See rndc(8) for details of the available rndc commands.

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

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

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

The key statement defines a key to be used by rndc when authenticating with named. Its syntax is identical to the key statement in named.conf. The keyword key is followed by a key name, which must be a valid domain name, though it need not actually be hierarchical; thus, a string like "rndc" is a valid name. The key statement has two clauses: algorithm and secret. While the configuration parser will accept any string as the argument to algorithm, currently only the string "hmac-md5" has any meaning. The secret is a base-64 encoded string as specified in RFC 3548.

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

A sample minimal configuration file is as follows:

key rndc_key {
   algorithm "hmac-md5";
   secret "c3Ryb25nIGVub3VnaCBmb3IgYSBtYW4gYnV0IG1hZGUgZm9yIGEgd29tYW4K";}
};

options {
   default-server 127.0.0.1;
   default-key rndc_key;
};

This file, if installed as /etc/rndc.conf, would allow the command:
$\texttt{rndc reload}

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

\begin{verbatim}
controls {
    inet 127.0.0.1 allow { localhost; } keys { rndc_key; };
};
\end{verbatim}

and it had an identical key statement for rndc_key.

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

### 3.3.2 Signals

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

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>Causes the server to read \texttt{named.conf} and reload the database.</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>Causes the server to clean up and exit.</td>
</tr>
<tr>
<td>SIGINT</td>
<td>Causes the server to clean up and exit.</td>
</tr>
</tbody>
</table>
Chapter 4

Advanced DNS Features

4.1 Notify

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

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

NOTE

As a slave zone can also be a master to other slaves, named, by default, sends NOTIFY messages for every zone it loads. Specifying notify master-only; will cause named to only send NOTIFY for master zones that it loads.

4.2 Dynamic Update

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

Dynamic update is enabled by including an allow-update or update-policy clause in the zone statement. The tkey-gssapi-credential and tkey-domain clauses in the options statement enable the server to negotiate keys that can be matched against those in update-policy or allow-update.

Updating of secure zones (zones using DNSSEC) follows RFC 3007: RRSIG, NSEC and NSEC3 records affected by updates are automatically regenerated by the server using an online zone key. Update authorization is based on transaction signatures and an explicit server policy.

4.2.1 The journal file

All changes made to a zone using dynamic update are stored in the zone’s journal file. This file is automatically created by the server when the first dynamic update takes place. The name of the journal file is formed by appending the extension .jnl to the name of the corresponding zone file unless specifically overridden. The journal file is in a binary format and should not be edited manually.
The server will also occasionally write ("dump") the complete contents of the updated zone to its zone file. This is not done immediately after each dynamic update, because that would be too slow when a large zone is updated frequently. Instead, the dump is delayed by up to 15 minutes, allowing additional updates to take place. During the dump process, transient files will be created with the extensions .jnw and .jbk; under ordinary circumstances, these will be removed when the dump is complete, and can be safely ignored.

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

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

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

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

### 4.3 Incremental Zone Transfers (IXFR)

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

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

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

### 4.4 Split DNS

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

One common reason for setting up a DNS system this way is to hide “internal” DNS information from “external” clients on the Internet. There is some debate as to whether or not this is actually useful. Internal DNS information leaks out in many ways (via email headers, for example) and most savvy “attackers” can find the information they need using other means. However, since listing addresses of internal servers that external clients cannot possibly reach can result in connection delays and other annoyances, an organization may choose to use a Split DNS to present a consistent view of itself to the outside world.

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

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

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

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

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

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

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

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

Here’s an example of a wildcard MX record:

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

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

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

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

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

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

Hosts on the Internet will be able to:

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

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

Internal DNS server config:
acl internals { 172.16.72.0/24; 192.168.1.0/24; }

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

options {
    ... ...
    forward only;
    forwarders { // forward to external servers
        bastion-ips-go-here;
    };
    allow-transfer { none; };
    allow-query { internals; externals; }; // restrict query access
    allow-recursion { internals; }; // restrict recursion
    ...
    ...
};

zone "site1.example.com" { // sample master zone
    type master;
    file "m/site1.example.com";
    forwarders { }; // do normal iterative
    allow-query { internals; externals; };
    allow-transfer { internals; };
};

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

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

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

External (bastion host) DNS server config:

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

acl externals { bastion-ips-go-here; }
4.5. TSIG

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

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

TSIG can also be useful for dynamic update. A primary server for a dynamic zone should control access to the dynamic update service, but IP-based access control is insufficient. The cryptographic access control provided by TSIG is far superior. The nsupdate program supports TSIG via the \(-k\) and \(-y\) command line options or inline by use of the \texttt{key}.

4.5.1 Generate Shared Keys for Each Pair of Hosts

A shared secret is generated to be shared between \textit{host1} and \textit{host2}. An arbitrary key name is chosen: "\textit{host1-host2}". The key name must be the same on both hosts.
4.5. TSIG

4.5.1.1 Automatic Generation

The following command will generate a 128-bit (16 byte) HMAC-SHA256 key as described above. Longer keys are better, but shorter keys are easier to read. Note that the maximum key length is the digest length, here 256 bits.

dnssec-keygen -a hmac-sha256 -b 128 -n HOST host1-host2.

The key is in the file Khost1-host2.+163+00000.private. Nothing directly uses this file, but the base-64 encoded string following "Key:" can be extracted from the file and used as a shared secret:

Key: La/E5CjG9O+os1jq0a2jdA==

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

4.5.1.2 Manual Generation

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

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

4.5.2 Copying the Shared Secret to Both Machines

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

4.5.3 Informing the Servers of the Key’s Existence

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

```
key host1-host2. {
    algorithm hmac-sha256;
    secret "La/E5CjG9O+os1jq0a2jdA==";
};
```

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

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

4.5.4 Instructing the Server to Use the Key

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

```
server 10.1.2.3 {
    keys { host1-host2. ;};
};
```
Multiple keys may be present, but only the first is used. This directive does not contain any secrets, so it may be in a world-readable file.

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

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

### 4.5.5 TSIG Key Based Access Control

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

An example of an allow-update directive would be:

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

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

You may want to read about the more powerful update-policy statement in Section 6.2.26.4.

### 4.5.6 Errors

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

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

### 4.6 TKEY

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

The TKEY process is initiated by a client or server by sending a signed TKEY query (including any appropriate KEYS) to a TKEY-aware server. The server response, if it indicates success, will contain a TKEY record and any appropriate keys. After this exchange, both participants have enough information to determine the shared secret; the exact process depends on the TKEY mode. When using the Diffie-Hellman TKEY mode, Diffie-Hellman keys are exchanged, and the shared secret is derived by both participants.
4.7 SIG(0)

BIND 9 partially supports DNSSEC SIG(0) transaction signatures as specified in RFC 2535 and RFC 2931. SIG(0) uses public/private keys to authenticate messages. Access control is performed in the same manner as TSIG keys; privileges can be granted or denied based on the key name.

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

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

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

4.8 DNSSEC

Cryptographic authentication of DNS information is possible through the DNS Security (DNSSEC-bis) extensions, defined in RFC 4033, RFC 4034, and RFC 4035. This section describes the creation and use of DNSSEC signed zones.

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

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

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

4.8.1 Generating Keys

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

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

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

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

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

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

The `dnssec-keyfromlabel` program is used to get a key pair from a crypto hardware and build the key files. Its usage is similar to `dnssec-keygen`.

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

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

Any keyset files corresponding to secure subzones should be present. The zone signer will generate NSEC, NSEC3 and RRSIG records for the zone, as well as DS for the child zones if ‘-g’ is specified. If ‘-g’ is not specified, then DS RRsets for the secure child zones need to be added manually.

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

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

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

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

4.8.3 Configuring Servers

To enable `named` to respond appropriately to DNS requests from DNSSEC aware clients, `dnssec-enable` must be set to yes. (This is the default setting.)

To enable `named` to validate answers from other servers, the `dnssec-enable` and `dnssec-validation` options must both be set to yes (the default setting in BIND 9.5 and later), and at least one trust anchor must be configured with a `trusted-keys` statement in `named.conf`.

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

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

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

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

```bash
trusted-keys {
  /* Root Key */
  "." 257 3 3 "BNY4wrWM1nCfJ+CXd0rVXYyMmobt7sEEfK3clRbGaTwSjxGkxJWo2u6I7PzJu/E9gx4UClzGAh1xKdE4zYipRhaBNKvcc2U9m2hkdUpd1Vsc/HAjJNe8LmMlnzY3zy2Xy4k1W0ADTPzSv9emj8V18PHgjBLAtVtvYyv/1n52ApjYghf+6f6LrmlKdazMQ2OcnACR817DF4BBA7UR/bDHy5iWTXWSi6XmoJLB9Gscq7l70KDqlvXR3M/1UUVRbkeg1IPSJsidmK32yC11h4XSKbje/45SKucHgnwU5jefMtq66gKodQj+MA21AUFUvE7u99WzTlZy3qIxHhxYQ2OQF975s+LKTpQcqx27R7AT3/V5hRQxScINqwc4jYqZ2DfDxgxbcDTC1U0CRbdiley1MNzxG3";

  /* Key for our organization's forward zone */
  example.com. 257 3 5 "AwEAaxPPMcR2x0HBoQV4WzB6oEDX+r0QM65KbhTjrW1ZaARMpHEZze3Y9ifgEUq7v/zGZUdECSNvY+jZjuS01UpwqGuhUS1558Hb4JKUb0jTcm8PwxFj0eIX3oDfVmh10444gLkbvUKl/fcM7HfY/Be2GmC1rnnKjrp04yzy9f0q3Mk7jbfW3jUKvTTh25G17G7QUsxb5Usxt8lGnyTUHStlt3JwC5hKz6CqFxMvAZP20igTitixin/1LCrgX/KMEGd/biuvF4jJCyuieHukuY3H4XMacR+xia2 nIUFPvm/oyWR8BW/hWdzOvn
```
4.9 IPv6 Support in BIND 9

BIND 9 fully supports all currently defined forms of IPv6 name to address and address to name lookups. It will also use IPv6 addresses to make queries when running on an IPv6 capable system.
For forward lookups, BIND 9 supports only AAAA records. RFC 3363 deprecated the use of A6 records, and client-side support for A6 records was accordingly removed from BIND 9. However, authoritative BIND 9 name servers still load zone files containing A6 records correctly, answer queries for A6 records, and accept zone transfer for a zone containing A6 records.

For IPv6 reverse lookups, BIND 9 supports the traditional “nibble” format used in the ip6.arpa domain, as well as the older, deprecated ip6.int domain. Older versions of BIND 9 supported the “binary label” (also known as “bitstring”) format, but support of binary labels has been completely removed per RFC 3363. Many applications in BIND 9 do not understand the binary label format at all any more, and will return an error if given. In particular, an authoritative BIND 9 name server will not load a zone file containing binary labels.

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

### 4.9.1 Address Lookups Using AAAA Records

The IPv6 AAAA record is a parallel to the IPv4 A record, and, unlike the deprecated A6 record, specifies the entire IPv6 address in a single record. For example,

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

Use of IPv4-in-IPv6 mapped addresses is not recommended. If a host has an IPv4 address, use an A record, not a AAAA, with :ffff:192.168.42.1 as the address.

### 4.9.2 Address to Name Lookups Using Nibble Format

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

```
$ORIGIN 0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0 14400 IN PTR host.example.com.
```
Chapter 5

The BIND 9 Lightweight Resolver

5.1 The Lightweight Resolver Library

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

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

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

5.2 Running a Resolver Daemon

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

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

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

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

The lwresd daemon may also be configured with a named.conf style configuration file, in /etc/lwresd.conf by default. A name server may also be configured to act as a lightweight resolver daemon using the lwres statement in named.conf.
Chapter 6

BIND 9 Configuration Reference

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

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

6.1 Configuration File Elements

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

<table>
<thead>
<tr>
<th>acl_name</th>
<th>The name of an address_match_list as defined by the acl statement.</th>
</tr>
</thead>
<tbody>
<tr>
<td>address_match_list</td>
<td>A list of one or more ip_addr, ip_prefix, key_id, or acl_name elements, see Section 6.1.1.</td>
</tr>
<tr>
<td>masters_list</td>
<td>A named list of one or more ip_addr with optional key_id and/or ip_port. A masters_list may include other masters_lists.</td>
</tr>
<tr>
<td>domain_name</td>
<td>A quoted string which will be used as a DNS name, for example &quot;my.test.domain&quot;.</td>
</tr>
<tr>
<td>dotted_decimal</td>
<td>One to four integers valued 0 through 255 separated by dots (&quot;.&quot;), such as 123, 45.67 or 89.123.45.67.</td>
</tr>
<tr>
<td>ip4_addr</td>
<td>An IPv4 address with exactly four elements in dotted_decimal notation.</td>
</tr>
<tr>
<td>ip6_addr</td>
<td>An IPv6 address, such as 2001:db8::1234. IPv6 scoped addresses that have ambiguity on their scope zones must be disambiguated by an appropriate zone ID with the percent character (&quot;%&quot;) as delimiter. It is strongly recommended to use string zone names rather than numeric identifiers, in order to be robust against system configuration changes. However, since there is no standard mapping for such names and identifier values, currently only interface names as link identifiers are supported, assuming one-to-one mapping between interfaces and links. For example, a link-local address fe80::1 on the link attached to the interface ne0 can be specified as fe80::1%ne0. Note that on most systems link-local addresses always have the ambiguity, and need to be disambiguated.</td>
</tr>
<tr>
<td>ip_addr</td>
<td>An ip4_addr or ip6_addr.</td>
</tr>
</tbody>
</table>

27
### 6.1. Configuration File Elements

#### Chapter 6. BIND 9 Configuration Reference

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ip_port</strong></td>
<td>An IP port number. The number is limited to 0 through 65535, with values below 1024 typically restricted to use by processes running as root. In some cases, an asterisk ('*') character can be used as a placeholder to select a random high-numbered port.</td>
</tr>
<tr>
<td><strong>ip_prefix</strong></td>
<td>An IP network specified as an ip_addr, followed by a slash ('/') and then the number of bits in the netmask. Trailing zeros in a ip_addr may be omitted. For example, 127/8 is the network 127.0.0.0 with netmask 255.0.0.0 and 1.2.3.0/28 is network 1.2.3.0 with netmask 255.255.255.240. When specifying a prefix involving a IPv6 scoped address the scope may be omitted. In that case the prefix will match packets from any scope.</td>
</tr>
<tr>
<td><strong>key_id</strong></td>
<td>A domain_name representing the name of a shared key, to be used for transaction security.</td>
</tr>
<tr>
<td><strong>key_list</strong></td>
<td>A list of one or more key_ids, separated by semicolons and ending with a semicolon.</td>
</tr>
<tr>
<td><strong>number</strong></td>
<td>A non-negative 32-bit integer (i.e., a number between 0 and 4294967295, inclusive). Its acceptable value might further be limited by the context in which it is used.</td>
</tr>
<tr>
<td><strong>path_name</strong></td>
<td>A quoted string which will be used as a pathname, such as zones/master/my.test.domain.</td>
</tr>
<tr>
<td><strong>port_list</strong></td>
<td>A list of an ip_port or a port range. A port range is specified in the form of range followed by two port_numbers, port_low and port_high, which represents port numbers from port_low through port_high, inclusive. port_low must not be larger than port_high. For example, range 1024 65535 represents ports from 1024 through 65535. In either case an asterisk ('*') character is not allowed as a valid ip_port.</td>
</tr>
<tr>
<td><strong>size_spec</strong></td>
<td>A 64-bit unsigned integer, or the keywords unlimited or default. Integers may take values $0 &lt;= value &lt;= 18446744073709551615$, though certain parameters may use a more limited range within these extremes. In most cases, setting a value to 0 does not literally mean zero; it means &quot;undefined&quot; or &quot;as big as possible&quot;, depending on the context. See the explanations of particular parameters that use size_spec for details on how they interpret its use. Numeric values 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<em>1024, and 1024</em>1024*1024 respectively. unlimited generally means &quot;as big as possible&quot;, though in certain contexts, (including max-cache-size), it may mean the largest possible 32-bit unsigned integer (0xffffffff); this distinction can be important when dealing with larger quantities. unlimited is usually the best way to safely set a very large number. default uses the limit that was in force when the server was started.</td>
</tr>
<tr>
<td><strong>yes_or_no</strong></td>
<td>Either yes or no. The words true and false are also accepted, as are the numbers 1 and 0.</td>
</tr>
<tr>
<td><strong>dialup_option</strong></td>
<td>One of yes, no, notify, notify-passive, refresh or passive. When used in a zone, notify-passive, refresh, and passive are restricted to slave and stub zones.</td>
</tr>
</tbody>
</table>
6.1.1 Address Match Lists

6.1.1.1 Syntax

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

6.1.1.2 Definition and Usage

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

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

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

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

When a given IP address or prefix is compared to an address match list, the comparison takes place in approximately O(1) time. However, key comparisons require that the list of keys be traversed until a matching key is found, and therefore may be somewhat slower.

The interpretation of a match depends on whether the list is being used for access control, defining listen-on ports, or in a sortlist, and whether the element was negated.

When used as an access control list, a non-negated match allows access and a negated match denies access. If there is no match, access is denied. The clauses allow-notify, allow-recursion, allow-recursion-on, allow-query, allow-query-on, allow-query-cache, allow-query-cache-on, allow-transfer, allow-update, allow-update-forwarding, and blackhole all use address match lists. Similarly, the listen-on option will cause the server to refuse queries on any of the machine’s addresses which do not match the list.

Order of insertion is significant. If more than one element in an ACL is found to match a given IP address or prefix, preference will be given to the one that came first in the ACL definition. Because of this first-match behavior, an element that defines a subset of another element in the list should come before the broader element, regardless of whether either is negated. For example, in 1.2.3/24; ! 1.2.3.13; the 1.2.3.13 element is completely useless because the algorithm will match any lookup for 1.2.3.13 to the 1.2.3/24 element. Using ! 1.2.3.13; 1.2.3/24 fixes that problem by having 1.2.3.13 blocked by the negation, but all other 1.2.3.* hosts fall through.

6.1.2 Comment Syntax

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

/* This is a BIND comment as in C */
// This is a BIND comment as in C++
# This is a BIND comment as in common UNIX shells and perl

6.1.2.2 Definition and Usage

Comments may appear anywhere that 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, as in C++ comments. For example:

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

WARNING

You cannot use the semicolon (';') character to start a comment such as you would in a zone file. The semicolon indicates the end of a configuration statement.

6.2 Configuration File Grammar

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

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

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

6.2.1 acl Statement Grammar

acl acl-name {
address_match_list
};

6.2.2 acl Statement Definition and Usage

The acl statement assigns a symbolic name to an address match list. It gets its name from a primary use of address match lists: Access Control Lists (ACLs).

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

The following ACLs are built-in:

<table>
<thead>
<tr>
<th>acl</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>Matches all hosts.</td>
</tr>
<tr>
<td>none</td>
<td>Matches no hosts.</td>
</tr>
<tr>
<td>localhost</td>
<td>Matches the IPv4 and IPv6 addresses of all network interfaces on the system.</td>
</tr>
<tr>
<td>localnets</td>
<td>Matches any host on an IPv4 or IPv6 network for which the system has an interface. Some systems do not provide a way to determine the prefix lengths of local IPv6 addresses. In such a case, localnets only matches the local IPv6 addresses, just like localhost.</td>
</tr>
</tbody>
</table>
6.2.3 controls Statement Grammar

controls {
  [ inet ( ip_addr | * ) [ port ip_port ] allow { address_match_list } 
    keys { key_list } ]; ]
  [ inet ...; ]
  [ unix path perm number owner number group number keys { key_list } ]; ]
  [ unix ...; ]
};

6.2.4 controls Statement Definition and Usage

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

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

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

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

A unix control channel is a UNIX domain socket listening at the specified path in the file system. Access to the socket is specified by the perm, owner and group clauses. Note on some platforms (SunOS and Solaris) the permissions (perm) are applied to the parent directory as the permissions on the socket itself are ignored.

The primary authorization mechanism of the command channel is the key_list, which contains a list of key_ids. Each key_id in the key_list is authorized to execute commands over the control channel. See [Remote Name Daemon Control application] in Section 3.3.1.2 for information about configuring keys in rndc.

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

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

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

To disable the command channel, use an empty controls statement: controls { };
6.2.5 include Statement Grammar

include filename;

6.2.6 include Statement Definition and Usage

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

6.2.7 key Statement Grammar

key key_id {
    algorithm string;
    secret string;
};

6.2.8 key Statement Definition and Usage

The key statement defines a shared secret key for use with TSIG (see Section 4.5) or the command channel (see Section 6.2.4). The key statement can occur at the top level of the configuration file or inside a view statement. Keys defined in top-level key statements can be used in all views. Keys intended for use in a controls statement (see Section 6.2.4) must be defined at the top level.

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

The algorithm_id is a string that specifies a security/authentication algorithm. Named supports hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384 and hmac-sha512 TSIG authentication. Truncated hashes are supported by appending the minimum number of required bits preceded by a dash, e.g. hmac-sha1-80. The secret_string is the secret to be used by the algorithm, and is treated as a base-64 encoded string.

6.2.9 logging Statement Grammar

logging {
[ channel channel_name {
   ( file path_name
       [ versions ( number | unlimited ) ]
       [ size size_spec ]
       | syslog syslog_facility
       | stderr
       | null );
   [ severity (critical | error | warning | notice |
                info | debug [ level ] | dynamic ); ]
   [ print-category yes or no; ]
   [ print-severity yes or no; ]
   [ print-time yes or no; ]
}; ]
[ category category_name {
6.2.10  logging Statement Definition and Usage

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

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

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

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

6.2.10.1 The channel Phrase

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

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

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

The `file` destination clause directs the channel to a disk file. It can include limitations both on how large the file is allowed to become, and how many versions of the file will be saved each time the file is opened.

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

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

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

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

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

On Windows machines syslog messages are directed to the EventViewer.

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

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

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

The server can supply extensive debugging information when it is in debugging mode. If the server’s global debug level is greater than zero, then debugging mode will be active. The global debug level is set either by starting the **named** server with the **-d** flag followed by a positive integer, or by running **rndc trace**. The global debug level can be set to zero, and debugging mode turned off, by running **rndc notrace**. All debugging messages in the server have a debug level, and higher debug levels give more detailed output. Channels that specify a specific debug severity, for example:

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

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

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

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

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

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

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

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

### 6.2.10.2 The category Phrase

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

```plaintext
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:

```plaintext
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:

```plaintext
category xfer-out { null; };
category notify { null; };
```
Following are the available categories and brief descriptions of the types of log information they contain. More categories may be added in future BIND releases.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>The default category defines the logging options for those categories where no specific configuration has been defined.</td>
</tr>
<tr>
<td>general</td>
<td>The catch-all. Many things still aren’t classified into categories, and they all end up here.</td>
</tr>
<tr>
<td>database</td>
<td>Messages relating to the databases used internally by the name server to store zone and cache data.</td>
</tr>
<tr>
<td>security</td>
<td>Approval and denial of requests.</td>
</tr>
<tr>
<td>config</td>
<td>Configuration file parsing and processing.</td>
</tr>
<tr>
<td>resolver</td>
<td>DNS resolution, such as the recursive lookups performed on behalf of clients by a caching name server.</td>
</tr>
<tr>
<td>xfer-in</td>
<td>Zone transfers the server is receiving.</td>
</tr>
<tr>
<td>xfer-out</td>
<td>Zone transfers the server is sending.</td>
</tr>
<tr>
<td>notify</td>
<td>The NOTIFY protocol.</td>
</tr>
<tr>
<td>client</td>
<td>Processing of client requests.</td>
</tr>
<tr>
<td>unmatched</td>
<td>Messages that named was unable to determine the class of or for which there was no matching view. A one line summary is also logged to the client category. This category is best sent to a file or stderr, by default it is sent to the null channel.</td>
</tr>
<tr>
<td>network</td>
<td>Network operations.</td>
</tr>
<tr>
<td>update</td>
<td>Dynamic updates.</td>
</tr>
<tr>
<td>update-security</td>
<td>Approval and denial of update requests.</td>
</tr>
<tr>
<td>queries</td>
<td>Specify where queries should be logged to.</td>
</tr>
<tr>
<td>query-errors</td>
<td>Information about queries that resulted in some failure.</td>
</tr>
<tr>
<td>dispatch</td>
<td>Dispatching of incoming packets to the server modules where they are to be processed.</td>
</tr>
<tr>
<td>dnssec</td>
<td>DNSSEC and TSIG protocol processing.</td>
</tr>
<tr>
<td>lame-servers</td>
<td>Lame servers. These are misconfigurations in remote servers, discovered by BIND 9 when trying to query those servers during resolution.</td>
</tr>
<tr>
<td>delegation-only</td>
<td>Delegation only. Logs queries that have been forced to NXDOMAIN as the result of a delegation-only zone or a delegation-only in a hint or stub zone declaration.</td>
</tr>
</tbody>
</table>
edns-disabled

Log queries that have been forced to use plain DNS due to timeouts. This is often due to the remote servers not being RFC 1034 compliant (not always returning FORMERR or similar to EDNS queries and other extensions to the DNS when they are not understood). In other words, this is targeted at servers that fail to respond to DNS queries that they don’t understand.

Note: the log message can also be due to packet loss. Before reporting servers for non-RFC 1034 compliance they should be re-tested to determine the nature of the non-compliance. This testing should prevent or reduce the number of false-positive reports.

Note: eventually named will have to stop treating such timeouts as due to RFC 1034 non compliance and start treating it as plain packet loss. Falsely classifying packet loss as due to RFC 1034 non compliance impacts on DNSSEC validation which requires EDNS for the DNSSEC records to be returned.

6.2.10.3 The query-errors Category

The query-errors category is specifically intended for debugging purposes: To identify why and how specific queries result in responses which indicate an error. Messages of this category are therefore only logged with debug levels.

At the debug levels of 1 or higher, each response with thercode of SERVFAIL is logged as follows:

client 127.0.0.1#61502: query failed (SERVFAIL) for www.example.com/IN/AAAA at query.c:3880

This means an error resulting in SERVFAIL was detected at line 3880 of source file query.c. Log messages of this level will particularly help identify the cause of SERVFAIL for an authoritative server.

At the debug levels of 2 or higher, detailed context information of recursive resolutions that resulted in SERVFAIL is logged. The log message will look like as follows:

fetch completed at resolver.c:2970 for www.example.com/A in 30.000183: timed out/success [domain:example.com,referral:2,restart:7,qrysent:8,timeout:5,lame:0,neterr:0,badresp:1,adberr:0,findfail:0,valfail:0]

The first part before the colon shows that a recursive resolution for AAAA records of www.example.com completed in 30.000183 seconds and the final result that led to the SERVFAIL was determined at line 2970 of source file resolver.c.

The following part shows the detected final result and the latest result of DNSSEC validation. The latter is always success when no validation attempt is made. In this example, this query resulted in SERVFAIL probably because all name servers are down or unreachable, leading to a timeout in 30 seconds. DNSSEC validation was probably not attempted.

The last part enclosed in square brackets shows statistics information collected for this particular resolution attempt. The domain field shows the deepest zone that the resolver reached; it is the zone where the error was finally detected. The meaning of the other fields is summarized in the following table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>referral</td>
<td>The number of referrals the resolver received throughout the resolution process</td>
</tr>
<tr>
<td>restart</td>
<td>The number of cycles that the resolver tried remote servers at the domain zone. In each cycle the resolver sent a query to each known name server of the domain zone</td>
</tr>
<tr>
<td>qrysent</td>
<td>The number of queries the resolver sent at the domain zone</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of timeouts since the resolver received the last response</td>
</tr>
<tr>
<td>lame</td>
<td>The number of lame servers the resolver detected at the domain zone. A server is detected to be lame either because it returned an invalid response or because the resolver found the server to be broken by looking up in BIND9’s address database (ADB), where lame servers are cached</td>
</tr>
<tr>
<td>neterr</td>
<td>The number of erroneous results that the resolver encountered in sending queries at the domain zone. One common case is the remote server is unreachable and the resolver receives an ICMP unreachable error message</td>
</tr>
<tr>
<td>badresp</td>
<td>The number of unexpected responses (other than lame) to queries sent by the resolver at the domain zone. In this example this is 1, which means that one unexpected response was not classified as lame or as an erroneous result</td>
</tr>
</tbody>
</table>

38
Failures in finding remote server addresses of the domain zone in the ADB. One common case of this is that the remote server's name does not have any address records.

Failures of resolving remote server addresses. This is a total number of failures throughout the resolution process.

Failures of DNSSEC validation. Validation failures are counted throughout the resolution process (not limited to the domain zone), but should only happen in domain.

At the debug levels of 3 or higher, the same messages as those at the debug 1 level are logged for other errors than SERVFAIL. Note that negative responses such as NXDOMAIN are not regarded as errors here.

At the debug levels of 4 or higher, the same messages as those at the debug 2 level are logged for other errors than SERVFAIL. Unlike the above case of level 3, messages are logged for negative responses. This is because any unexpected results can be difficult to debug in the recursion case.

### 6.2.11 lwres Statement Grammar

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

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

### 6.2.12 lwres Statement Definition and Usage

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

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

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

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

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

### 6.2.13 masters Statement Grammar

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

### 6.2.14 masters Statement Definition and Usage

`masters` lists allow for a common set of masters to be easily used by multiple stub and slave zones.
6.2.15 options Statement Grammar

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

```plaintext
options {
  [ version version_string; ]
  [ hostname hostname_string; ]
  [ server-id server_id_string; ]
  [ directory path_name; ]
  [ key-directory path_name; ]
  [ named-xfer path_name; ]
  [ tkey-gssapi-credential principal; ]
  [ tkey-domain domainname; ]
  [ tkey-dhkey key_name key_tag; ]
  [ cache-file path_name; ]
  [ dump-file path_name; ]
  [ memstatistics yes_or_no; ]
  [ memstatistics-file path_name; ]
  [ pid-file path_name; ]
  [ recursing-file path_name; ]
  [ statistics-file path_name; ]
  [ zone-statistics yes_or_no; ]
  [ auth-nxdomain yes_or_no; ]
  [ deallocate-on-exit yes_or_no; ]
  [ dialup dialup_option; ]
  [ fake-iquery yes_or_no; ]
  [ fetch-glue yes_or_no; ]
  [ flush-zones-on-shutdown yes_or_no; ]
  [ has-old-clients yes_or_no; ]
  [ host-statistics yes_or_no; ]
  [ host-statistics-max number; ]
  [ minimal-responses yes_or_no; ]
  [ multiple-cnames yes_or_no; ]
  [ notify yes_or_no | explicit | master-only; ]
  [ recursion yes_or_no; ]
  [ request-naid yes_or_no; ]
  [ rfc2308-type1 yes_or_no; ]
  [ use-id-pool yes_or_no; ]
  [ maintain-ixfr-base yes_or_no; ]
  [ ixfr-from-differences (yes_or_no | master | slave); ]
  [ dnssec-enable yes_or_no; ]
  [ dnssec-validation yes_or_no; ]
  [ dnssec-lookaside domain trust-anchor domain; ]
  [ dnssec-must-be-secure domain yes_or_no; ]
  [ dnssec-accept-expired yes_or_no; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr [port ip_port] ; ... ] } ]; ]
  [ dual-stack-servers [port ip_port] ]
    ( domain_name [port ip_port] | ip_addr [port ip_port] ) ;
    ... ); ]
  [ check-names ( master | slave | response )
    ( warn | fail | ignore ); ]
  [ check-mx ( warn | fail | ignore ); ]
  [ check-wildcard yes_or_no; ]
  [ check-integrity yes_or_no; ]
  [ check-mx-cname ( warn | fail | ignore ); ]
  [ check-srv-cname ( warn | fail | ignore ); ]
  [ check-sibling yes_or_no; ]
}
[ check-spf { warn | fail | ignore ); ]
[ allow-notify { address_match_list } ];
[ allow-query { address_match_list } ];
[ allow-query-on { address_match_list } ];
[ allow-query-cache { address_match_list } ];
[ allow-transfer { address_match_list } ];
[ allow-recursion { address_match_list } ];
[ allow-recursion-on { address_match_list } ];
[ allow-update { address_match_list } ];
[ allow-update-forwarding { address_match_list } ];
[ update-check-ksk yes_or_no; ]
[ try-tcp-refresh yes_or_no; ]
[ allow-v6-synthesis { address_match_list } ];
[ blackhole { address_match_list } ];
[ use-v4-udp-ports { port_list } ];
[ avoid-v4-udp-ports { port_list } ];
[ use-v6-udp-ports { port_list } ];
[ avoid-v6-udp-ports { port_list } ];
[ listen-on [ port ip_port ] { address_match_list } ];
[ listen-on-v6 [ port ip_port ] { address_match_list } ];
[ query-source ( ( ip4_addr | * ) [ port ( ip_port | * ) ] | [ address ( ip4_addr | * ) ] [ port ( ip_port | * ) ] ) ];
[ query-source-v6 ( ( ip6_addr | * ) [ port ( ip_port | * ) ] | [ address ( ip6_addr | * ) ] [ port ( ip_port | * ) ] ) ];
[ use-queryport-pool yes_or_no; ]
[ queryport-pool-ports number ];
[ queryport-pool-updateinterval number ];
[ max-transfer-time-in number ];
[ max-transfer-time-out number ];
[ max-transfer-time-out number ];
[ max-transfer-idle-in number ];
[ max-transfer-idle-out number ];
[ tcp-clients number ];
[ reserved-sockets number ];
[ recursive-clients number ];
[ serial-query-rate number ];
[ serial-queries number ];
[ tcp-listen-queue number ];
[ transfer-format ( one-answer | many-answers ); ]
[ transfers-in number ];
[ transfers-out number ];
[ transfers-per-ns number ];
[ transfer-source ( ip4_addr | * ) [ port ip_port ] ];
[ transfer-source-v6 ( ip6_addr | * ) [ port ip_port ] ];
[ alt-transfer-source ( ip4_addr | * ) [ port ip_port ] ];
[ alt-transfer-source-v6 ( ip6_addr | * ) [ port ip_port ] ];
[ use-alt-transfer-source yes_or_no ];
[ notify-delay seconds ];
[ notify-source ( ip4_addr | * ) [ port ip_port ] ];
[ notify-source-v6 ( ip6_addr | * ) [ port ip_port ] ];
[ notify-to-soa yes_or_no ];
[ also-notify { ip_addr [ port ip_port ] | ip_addr [ port ip_port ] ; ... } ];
[ max-ixfr-log-size number ];
[ max-journal-size size_spec ];
[ coresize size_spec ];
6.2.16  options Statement Definition and Usage

The options statement sets up global options to be used by BIND. This statement may appear only once in a configuration file. If there is no options statement, an options block with each option set to its default will be used.
directory The working directory of the server. Any non-absolute pathnames in the configuration file will be taken as relative to this directory. The default location for most server output files (e.g. named.run) is this directory. If a directory is not specified, the working directory defaults to ".", the directory from which the server was started. The directory specified should be an absolute path.

key-directory When performing dynamic update of secure zones, the directory where the public and private DNSSEC key files should be found, if different than the current working directory. The directory specified must be an absolute path. (Note that this option has no effect on the paths for files containing non-DNSSEC keys such as the rndc.key.

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

tkey-gssapi-credential The security credential with which the server should authenticate keys requested by the GSS-TSIG protocol. Currently only Kerberos 5 authentication is available and the credential is a Kerberos principal which the server can acquire through the default system key file, normally /etc/krb5.keytab. Normally this principal is of the form "DNS/server.domain". To use GSS-TSIG, tkey-domain must also be set.

tkey-domain The domain appended to the names of all shared keys generated with TKEY. When a client requests a TKEY exchange, it may or may not specify the desired name for the key. If present, the name of the shared key will be client specified part + tkey-domain. Otherwise, the name of the shared key will be random hex digits + tkey-domain. In most cases, the domainname should be the server’s domain name, or an otherwise non-existent subdomain like "tkey.domainname". If you are using GSS-TSIG, this variable must be defined.

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

cache-file This is for testing only. Do not use.

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

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

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

recurring-file The pathname of the file the server dumps the queries that are currently recursing when instructed to do so with rndc recursing. If not specified, the default is named.recurring.

statistics-file The pathname of the file the server appends statistics to when instructed to do so using rndc stats. If not specified, the default is named.stats in the server’s current directory. The format of the file is described in Section 6.4.0.1.
**port** The UDP/TCP port number the server uses for receiving and sending DNS protocol traffic. The default is 53. This option is mainly intended for server testing; a server using a port other than 53 will not be able to communicate with the global DNS.

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

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

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

DS queries are expected to be made to and be answered by delegation only zones. Such queries and responses are treated as an exception to delegation-only processing and are not converted to NXDOMAIN responses provided a CNAME is not discovered at the query name.

If a delegation only zone server also serves a child zone it is not always possible to determine whether a answer comes from the delegation only zone or the child zone. SOA NS and DNSKEY records are apex only records and a matching response that contains these records or DS is treated as coming from a child zone. RRSIG records are also examined to see if they are signed by a child zone or not. The authority section is also examined to see if there is evidence that the answer is from the child zone. Answers that are determined to be from a child zone are not converted to NXDOMAIN responses. Despite all these checks there is still a possibility of false negatives when a child zone is being served.

Similarly false positives can arise from empty nodes (no records at the name) in the delegation only zone when the query type is not ANY.

Note some TLDs are not delegation only (e.g. “DE”, “LV”, “US” and “MUSEUM”). This list is not exhaustive.

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

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

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

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

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

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

**memstatistics** Write memory statistics to the file specified by *memstatistics-file* at exit. The default is *no* unless ‘-m record’ is specified on the command line in which case it is *yes*.

**dialup** If *yes*, then the server treats all zones as if they are doing zone transfers across a dial-on-demand dialup link, which can be brought up by traffic originating from this server. This has different effects according to zone type and concentrates the zone maintenance so that it all happens in a short interval, once every *heartbeat-interval* and hopefully during the one call. It also suppresses some of the normal zone maintenance traffic. The default is *no*.

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

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

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

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

<table>
<thead>
<tr>
<th>dialup mode</th>
<th>normal refresh</th>
<th>heart-beat refresh</th>
<th>heart-beat notify</th>
</tr>
</thead>
<tbody>
<tr>
<td>no (default)</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>notify</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>refresh</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>passive</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>notify-passive</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

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

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

**flush-zones-on-shutdown** When the nameserver exits due receiving SIGTERM, flush or do not flush any pending zone writes. The default is **flush-zones-on-shutdown no**.
has-old-clients This option was incorrectly implemented in BIND 8, and is ignored by BIND 9. To achieve the intended effect of has-old-clients yes, specify the two separate options auth-nxdomain yes and rfc2308-type1 no instead.

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

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

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

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

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

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

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

notify-to-soa If yes do not check the nameservers in the NS RRset against the SOA MNAME. Normally a NOTIFY message is not sent to the SOA MNAME (SOA ORIGIN) as it is supposed to contain the name of the ultimate master. Sometimes, however, a slave is listed as the SOA MNAME in hidden master configurations and in that case you would want the ultimate master to still send NOTIFY messages to all the nameservers listed in the NS RRset.

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

request-nsid If yes, then an empty EDNS(0) NSID (Name Server Identifier) option is sent with all queries to authoritative name servers during iterative resolution. If the authoritative server returns an NSID option in its response, then its contents are logged in the resolver category at level info. The default is no.

rfc2308-type1 Setting this to yes will cause the server to send NS records along with the SOA record for negative answers. The default is no.
use-id-pool This option is obsolete. BIND 9 always allocates query IDs from a pool.

zone-statistics If yes, the server will collect statistical data on all zones (unless specifically turned off on a per-zone basis by specifying zone-statistics no in the zone statement). These statistics may be accessed using rndc stats, which will dump them to the file listed in the statistics-file. See also Section 6.4.0.1.

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

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

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

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

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

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

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

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

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

When a name server is non-recursively queried for a name that is not below the apex of any served zone, it normally answers with an "upwards referral" to the root servers or the servers of some other known parent of the query name. Since the data in an upwards referral comes from the cache, the server will not be able to provide upwards referrals when additional-from-cache no has been specified. Instead, it will respond to such queries with REFUSED. This should not cause any problems since upwards referrals are not required for the resolution process.
match-mapped-addresses If yes, then an IPv4-mapped IPv6 address will match any address match list entries that match the corresponding IPv4 address.

This option was introduced to work around a kernel quirk in some operating systems that causes IPv4 TCP connections, such as zone transfers, to be accepted on an IPv6 socket using mapped addresses. This caused address match lists designed for IPv4 to fail to match. However, named now solves this problem internally. The use of this option is discouraged.

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

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

ixfr-from-differences also accepts master and slave at the view and options levels which causes ixfr-from-differences to be enabled for all master or slave zones respectively. It is off by default.

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

dnssec-enable Enable DNSSEC support in named. Unless set to yes, named behaves as if it does not support DNSSEC. The default is yes.

dnssec-validation Enable DNSSEC validation in named. Note dnssec-enable also needs to be set to yes to be effective. The default is yes.

dnssec-accept-expired Accept expired signatures when verifying DNSSEC signatures. The default is no. Setting this option to “yes” leaves named vulnerable to replay attacks.

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

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

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

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

check-mx Check whether the MX record appears to refer to a IP address. The default is to warn. Other possible values are fail and ignore.
check-wildcard  This option is used to check for non-terminal wildcards. The use of non-terminal wildcards is almost always as a result of a failure to understand the wildcard matching algorithm (RFC 1034). This option affects master zones. The default (yes) is to check for non-terminal wildcards and issue a warning.

check-integrity  Perform post load zone integrity checks on master zones. This checks that MX and SRV records refer to address (A or AAAA) records and that glue address records exist for delegated zones. For MX and SRV records only in-zone hostnames are checked (for out-of-zone hostnames use named-checkzone). For NS records only names below top of zone are checked (for out-of-zone names and glue consistency checks use named-checkzone). The default is yes.

Check that the two forms of Sender Policy Framework records (TXT records starting with “v=spf1” and SPF) either both exist or both don’t exist. Warnings are emitted if they don’t and be suppressed with check-spf.

check-mx-cname  If check-integrity is set then fail, warn or ignore MX records that refer to CNAMEs. The default is to warn.

check-srv-cname  If check-integrity is set then fail, warn or ignore SRV records that refer to CNAMEs. The default is to warn.

check-sibling  When performing integrity checks, also check that sibling glue exists. The default is yes.

check-spf  When performing integrity checks, check that the two forms of Sender Policy Framework records (TXT records starting with “v=spf1” and SPF) both exist or both don’t exist and issue a warning if not met. The default is warn.

zero-no-soa-ttl  When returning authoritative negative responses to SOA queries set the TTL of the SOA record returned in the authority section to zero. The default is yes.

zero-no-soa-ttl-cache  When caching a negative response to a SOA query set the TTL to zero. The default is no.

update-check-ksk  When regenerating the RRSIGs following a UPDATE request to a secure zone, check the KSK flag on the DNSKEY RR to determine if this key should be used to generate the RRSIG. This flag is ignored if there are not DNSKEY RRs both with and without a KSK. The default is yes.

try-tcp-refresh  Try to refresh the zone using TCP if UDP queries fail. For BIND 8 compatibility, the default is yes.

6.2.16.2 Forwarding

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

forward  This option is only meaningful if the forwarders list is not empty. A value of first, the default, causes the server to query the forwarders first — and if that doesn’t answer the question, the server will then look for the answer itself. If only is specified, the server will only query the forwarders.
forwarders Specifies the IP addresses to be used for forwarding. The default is the empty list (no forwarding).

Forwarding can also be configured on a per-domain basis, allowing for the global forwarding options to be overridden in a variety of ways. You can set particular domains to use different forwarders, or have a different forward only/first behavior, or not forward at all, see Section 6.2.25.

6.2.16.3 Dual-stack Servers

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

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

6.2.16.4 Access Control

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

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

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

allow-query-on Specifies which local addresses can accept ordinary DNS questions. This makes it possible, for instance, to allow queries on internal-facing interfaces but disallow them on external-facing ones, without necessarily knowing the internal network’s addresses.

Note that allow-query-on is only checked for queries that are permitted by allow-query. A query must be allowed by both ACLs, or it will be refused.

allow-query-on may also be specified in the zone statement, in which case it overrides the options allow-query-on statement.

If not specified, the default is to allow queries on all addresses.

NOTE

allow-query-cache is now used to specify access to the cache.

allow-query-on

Specifies which local addresses can accept ordinary DNS questions. This makes it possible, for instance, to allow queries on internal-facing interfaces but disallow them on external-facing ones, without necessarily knowing the internal network’s addresses.

Note that allow-query-on is only checked for queries that are permitted by allow-query. A query must be allowed by both ACLs, or it will be refused.

allow-query-on may also be specified in the zone statement, in which case it overrides the options allow-query-on statement.

If not specified, the default is to allow queries on all addresses.
allow-query-cache is used to specify access to the cache.

allow-query-cache Specifies which hosts are allowed to get answers from the cache. If allow-query-cache is not set then allow-recursion is used if set, otherwise allow-query is used if set unless recursion no; is set in which case none; is used, otherwise the default (localnets; localhost) is used.

allow-query-cache-on Specifies which local addresses can give answers from the cache. If not specified, the default is to allow cache queries on any address, localnets and localhost.

allow-recursion Specifies which hosts are allowed to make recursive queries through this server. If allow-recursion is not set then allow-query-cache is used if set, otherwise allow-query is used if set, otherwise the default (localnets; localhost) is used.

allow-recursion-on Specifies which local addresses can accept recursive queries. If not specified, the default is to allow recursive queries on all addresses.

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

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

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

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

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

blackhole Specifies a list of addresses that the server will not accept queries from or use to resolve a query. Queries from these addresses will not be responded to. The default is none.

6.2.16.5 Interfaces

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

Multiple listen-on statements are allowed. For example,
listen-on { 5.6.7.8; };
listen-on port 1234 { !1.2.3.4; 1.2/16; };

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

If no \texttt{listen-on} is specified, the server will listen on port 53 on all IPv4 interfaces.

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

When
\begin{verbatim}
{ any; }
\end{verbatim}
is specified as the \texttt{address_match_list} for the \texttt{listen-on-v6} option, the server does not bind a separate
socket to each IPv6 interface address as it does for IPv4 if the operating system has enough API support
for IPv6 (specifically if it conforms to RFC 3493 and RFC 3542). Instead, it listens on the IPv6 wildcard
address. If the system only has incomplete API support for IPv6, however, the behavior is the same as
that for IPv4.

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

Multiple \texttt{listen-on-v6} options can be used. For example,
\begin{verbatim}
listen-on-v6 { any; };
listen-on-v6 port 1234 { !2001:db8::/32; any; };
\end{verbatim}

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

To make the server not listen on any IPv6 address, use
\begin{verbatim}
listen-on-v6 { none; };
\end{verbatim}

If no \texttt{listen-on-v6} option is specified, the server will not listen on any IPv6 address unless \texttt{-6} is specified
when \texttt{named} is invoked. If \texttt{-6} is specified then \texttt{named} will listen on port 53 on all IPv6 interfaces by
default.

\subsection{6.2.16.6 Query Address}

If the server doesn’t know the answer to a question, it will query other name servers. \texttt{query-source}
specifies the address and port used for such queries. For queries sent over IPv6, there is a separate
\texttt{query-source-v6} option. If \texttt{address} is * (asterisk) or is omitted, a wildcard IP address (INADDR\_ANY)
will be used.

If \texttt{port} is * or is omitted, a random port number from a pre-configured range is picked up and will be
used for each query. The port range(s) is that specified in the \texttt{use-v4-udp-ports} (for IPv4) and \texttt{use-v6-udp-ports}
(for IPv6) options, excluding the ranges specified in the \texttt{avoid-v4-udp-ports} and \texttt{avoid-v6-udp-ports}
options, respectively.

The defaults of the \texttt{query-source} and \texttt{query-source-v6} options are:
\begin{verbatim}
query-source address * port *;
query-source-v6 address * port *;
\end{verbatim}

If \texttt{use-v4-udp-ports} or \texttt{use-v6-udp-ports} is unspecified, \texttt{named} will check if the operating system pro-
vides a programming interface to retrieve the system’s default range for ephemeral ports. If such an
interface is available, \texttt{named} will use the corresponding system default range; otherwise, it will use its
own defaults:
use-v4-udp-ports { range 1024 65535; };
use-v6-udp-ports { range 1024 65535; };

Note: make sure the ranges be sufficiently large for security. A desirable size depends on various parameters, but we generally recommend it contain at least 16384 ports (14 bits of entropy). Note also that the system’s default range when used may be too small for this purpose, and that the range may even be changed while named is running; the new range will automatically be applied when named is reloaded. It is encouraged to configure use-v4-udp-ports and use-v6-udp-ports explicitly so that the ranges are sufficiently large and are reasonably independent from the ranges used by other applications.

Note: the operational configuration where named runs may prohibit the use of some ports. For example, UNIX systems will not allow named running without a root privilege to use ports less than 1024. If such ports are included in the specified (or detected) set of query ports, the corresponding query attempts will fail, resulting in resolution failures or delay. It is therefore important to configure the set of ports that can be safely used in the expected operational environment.

The defaults of the avoid-v4-udp-ports and avoid-v6-udp-ports options are:

avoid-v4-udp-ports {};
avoid-v6-udp-ports {};

Note: BIND 9.5.0 introduced the use-queryport-pool option to support a pool of such random ports, but this option is now obsolete because reusing the same ports in the pool may not be sufficiently secure. For the same reason, it is generally strongly discouraged to specify a particular port for the query-source or query-source-v6 options; it implicitly disables the use of randomized port numbers.

use-queryport-pool This option is obsolete.

queryport-pool-ports This option is obsolete.

queryport-pool-updateinterval This option is obsolete.

**Note**

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

**Note**

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

**Note**

- See also transfer-source and notify-source.
6.2. CONFIGURATION FILE GRAMMAR

CHAPTER 6. BIND 9 CONFIGURATION REFERENCE

6.2.16.7 Zone Transfers

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

also-notify Defines a global list of IP addresses of name servers that are also sent NOTIFY messages whenever a fresh copy of the zone is loaded, in addition to the servers listed in the zone’s NS records. This helps to ensure that copies of the zones will quickly converge on stealth servers. Optionally, a port may be specified with each also-notify address to send the notify messages to a port other than the default of 53. If an also-notify list is given in a zone statement, it will override the options also-notify statement. When a zone notify statement is set to no, the IP addresses in the global also-notify list will not be sent NOTIFY messages for that zone. The default is the empty list (no global notification list).

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

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

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

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

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

In addition to controlling the rate SOA refresh queries are issued at serial-query-rate also controls the rate at which NOTIFY messages are sent from both master and slave zones.

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

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

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

transfers-out The maximum number of outbound zone transfers that can be running concurrently. Zone transfer requests in excess of the limit will be refused. The default value is 10.
transfers-per-ns  The maximum number of inbound zone transfers that can be concurrently transferring from a given remote name server. The default value is 2. Increasing transfers-per-ns may speed up the convergence of slave zones, but it also may increase the load on the remote name server. transfers-per-ns may be overridden on a per-server basis by using the transfers phrase of the server statement.

transfer-source  transfer-source determines which local address will be bound to IPv4 TCP connections used to fetch zones transferred inbound by the server. It also determines the source IPv4 address, and optionally the UDP port, used for the refresh queries and forwarded dynamic updates. If not set, it defaults to a system controlled value which will usually be the address of the interface “closest to” the remote end. This address must appear in the remote end’s allow-transfer option for the zone being transferred, if one is specified. This statement sets the transfer-source for all zones, but can be overridden on a per-view or per-zone basis by including a transfer-source statement within the view or zone block in the configuration file.

NOTE

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

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

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

NOTE

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

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

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

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

use-v4-udp-ports, avoid-v4-udp-ports, use-v6-udp-ports, and avoid-v6-udp-ports specify a list of IPv4 and IPv6 UDP ports that will be used or not used as source ports for UDP messages. See Section 6.2.16.6 about how the available ports are determined. For example, with the following configuration

use-v6-udp-ports { range 32768 65535; };
avoid-v6-udp-ports { 40000; range 50000 60000; };

UDP ports of IPv6 messages sent from named will be in one of the following ranges: 32768 to 39999, 40001 to 49999, and 60001 to 65535.

avoid-v4-udp-ports and avoid-v6-udp-ports can be used to prevent named from choosing as its random source port a port that is blocked by your firewall or a port that is used by other applications; if a query went out with a source port blocked by a firewall, the answer would not get by the firewall and the name server would have to query again. Note: the desired range can also be represented only with use-v4-udp-ports and use-v6-udp-ports, and the avoid- options are redundant in that sense; they are provided for backward compatibility and to possibly simplify the port specification.

6.2.16.9 Operating System Resource Limits

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

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

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

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

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

stacksize The maximum amount of stack memory the server may use. The default is default.
6.2.16.10 Server Resource Limits

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

**max-ixfr-log-size** This option is obsolete; it is accepted and ignored for BIND 8 compatibility. The option **max-journal-size** performs a similar function in BIND 9.

**max-journal-size** Sets a maximum size for each journal file (see Section 4.2.1). When the journal file approaches the specified size, some of the oldest transactions in the journal will be automatically removed. The default is *unlimited*. This may also be set on a per-zone basis.

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

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

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

**reserved-sockets** The number of file descriptors reserved for TCP, stdio, etc. This needs to be big enough to cover the number of interfaces **named** listens on, **tcp-clients** as well as to provide room for outgoing TCP queries and incoming zone transfers. The default is *512*. The minimum value is *128* and the maximum value is *128* less than maxsockets (-S). This option may be removed in the future.

This option has little effect on Windows.

**max-cache-size** The maximum amount of memory to use for the server’s cache, in bytes. When the amount of data in the cache reaches this limit, the server will cause records to expire prematurely based on an LRU based strategy so that the limit is not exceeded. A value of 0 is special, meaning that records are purged from the cache only when their TTLs expire. Another special keyword **unlimited** means the maximum value of 32-bit unsigned integers (0xffffffff), which may not have the same effect as 0 on machines that support more than 32 bits of memory space. Any positive values less than 2MB will be ignored reset to 2MB. In a server with multiple views, the limit applies separately to the cache of each view. The default is 0.

**tcp-listen-queue** The listen queue depth. The default and minimum is 10. If the kernel supports the accept filter “dataready” this also controls how many TCP connections that will be queued in kernel space waiting for some data before being passed to accept. Nonzero values less than 10 will be silently raised. A value of 0 may also be used; on most platforms this sets the listen queue length to a system-defined default value.

6.2.16.11 Periodic Task Intervals

**cleaning-interval** This interval is effectively obsolete. Previously, the server would remove expired resource records from the cache every **cleaning-interval** minutes. BIND 9 now manages cache memory in a more sophisticated manner and does not rely on the periodic cleaning any more. Specifying this option therefore has no effect on the server’s behavior.
heartbeat-interval  The server will perform zone maintenance tasks for all zones marked as dialup whenever this interval expires. The default is 60 minutes. Reasonable values are up to 1 day (1440 minutes). The maximum value is 28 days (40320 minutes). If set to 0, no zone maintenance for these zones will occur.

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

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

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not yet implemented in BIND 9.</td>
</tr>
</tbody>
</table>

6.2.16.12  Topology

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

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

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

The default topology is

topology { localhost; localnets; };

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The topology option is not implemented in BIND 9.</td>
</tr>
</tbody>
</table>
The response to a DNS query may consist of multiple resource records (RRs) forming a resource records set (RRset). The name server will normally return the RRs within the RRset in an indeterminate order (but see the \texttt{rrset-order} statement in Section 6.2.16.14). The client resolver code should rearrange the RRs as appropriate, that is, using any addresses on the local net in preference to other addresses. However, not all resolvers can do this or are correctly configured. When a client is using a local server, the sorting can be performed in the server, based on the client’s address. This only requires configuring the name servers, not all the clients.

The \texttt{sortlist} statement (see below) takes an \texttt{address_match_list} and interprets it even more specifically than the \texttt{topology} statement does (Section 6.2.16.12). Each top level statement in the \texttt{sortlist} must itself be an explicit \texttt{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 \texttt{address_match_list}) of each top level list is checked against the source address of the query until a match is found.

Once the source address of the query has been matched, if the top level statement contains only one element, the actual primitive element that matched the source address is used to select the address in the response to move to the beginning of the response. If the statement is a list of two elements, then the second element is treated the same as the \texttt{address_match_list} in a \texttt{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.

\begin{verbatim}
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;
  { 192.168.2/24;
    192.168.3/24; }; };);
{ 192.168.2/24;
  { 192.168.2/24;
    192.168.3/24; }; };);
{ 192.168.3/24;
  { 192.168.1/24;
    192.168.3/24; }; };);
{ { 192.168.4/24; 192.168.5/24; }; // if .4 or .5, prefer that net }
};
\end{verbatim}

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

\begin{verbatim}
sortlist {
{ localhost; localnets; };  // IF the local host
  { localnets; };  // THEN first fit on the
};
\end{verbatim}
6.2.16.14  RRset Ordering

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

An order_spec is defined as follows:

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

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

The legal values for `ordering` are:

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

If BIND is configured with the "–enable-fixed-rrset" option at compile time, then the initial ordering of the RRset will match the one specified in the zone file.

For example:

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

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

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

**NOTE**

In this release of BIND 9, the `rrset-order` statement does not support "fixed" ordering by default. Fixed ordering can be enabled at compile time by specifying "–enable-fixed-rrset" on the "configure" command line.

6.2.16.15  Tuning

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

Lame-ttl also controls the amount of time DNSSEC validation failures are cached. There is a minimum of 30 seconds applied to bad cache entries if the lame-ttl is set to less than 30 seconds.

`max-ncache-ttl` To reduce network traffic and increase performance, the server stores negative answers. `max-ncache-ttl` is used to set a maximum retention time for these answers in the server in seconds. The default `max-ncache-ttl` is 10800 seconds (3 hours). `max-ncache-ttl` cannot exceed 7 days and will be silently truncated to 7 days if set to a greater value.
max-cache-ttl Sets the maximum time for which the server will cache ordinary (positive) answers. The default is one week (7 days). A value of zero may cause all queries to return SERVFAIL, because of lost caches of intermediate RRsets (such as NS and glue AAAA/A records) in the resolution process.

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

**NOTE**

Not implemented in BIND 9.

sig-validity-interval Specifies the number of days into the future when DNSSEC signatures automatically generated as a result of dynamic updates (Section 4.2) will expire. There is an optional second field which specifies how long before expiry that the signatures will be regenerated. If not specified, the signatures will be regenerated at 1/4 of base interval. The second field is specified in days if the base interval is greater than 7 days otherwise it is specified in hours. The default base interval is 30 days giving a re-signing interval of 7 1/2 days. The maximum values are 10 years (3660 days).

The signature inception time is unconditionally set to one hour before the current time to allow for a limited amount of clock skew.

The sig-validity-interval should be, at least, several multiples of the SOA expire interval to allow for reasonable interaction between the various timer and expiry dates.

sig-signing-nodes Specify the maximum number of nodes to be examined in each quantum when signing a zone with a new DNSKEY. The default is 100.

sig-signing-signatures Specify a threshold number of signatures that will terminate processing a quantum when signing a zone with a new DNSKEY. The default is 10.

sig-signing-type Specify a private RDATA type to be used when generating key signing records. The default is 65534.

It is expected that this parameter may be removed in a future version once there is a standard type.

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

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

The following defaults apply. **min-refresh-time** 300 seconds, **max-refresh-time** 2419200 seconds (4 weeks), **min-retry-time** 500 seconds, and **max-retry-time** 1209600 seconds (2 weeks).

edns-udp-size Sets the advertised EDNS UDP buffer size in bytes to control the size of packets received.

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

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

**masterfile-format** Specifies the file format of zone files (see Section 6.3.7). The default value is `text`, which is the standard textual representation. Files in other formats than `text` are typically expected to be generated by the `named-compilezone` tool. Note that when a zone file in a different format than `text` is loaded, named may omit some of the checks which would be performed for a file in the `text` format. In particular, check-names checks do not apply for the `raw` format. This means a zone file in the `raw` format must be generated with the same check level as that specified in the named configuration file. This statement sets the `masterfile-format` for all zones, but can be overridden on a per-zone or per-view basis by including a `masterfile-format` statement within the `zone` or `view` block in the configuration file.

**clients-per-query, max-clients-per-query** These set the initial value (minimum) and maximum number of recursive simultaneous clients for any given query (`<qname,qtype,qclass>`) that the server will accept before dropping additional clients. named will attempt to self-tune this value and changes will be logged. The default values are 10 and 100.

This value should reflect how many queries come in for a given name in the time it takes to resolve that name. If the number of queries exceed this value, named will assume that it is dealing with a non-responsive zone and will drop additional queries. If it gets a response after dropping queries, it will raise the estimate. The estimate will then be lowered in 20 minutes if it has remained unchanged.

If `clients-per-query` is set to zero, then there is no limit on the number of clients per query and no queries will be dropped.

If `max-clients-per-query` is set to zero, then there is no upper bound other than imposed by `recursive-clients`.

**notify-delay** The delay, in seconds, between sending sets of notify messages for a zone. The default is five (5) seconds.

The overall rate that NOTIFY messages are sent for all zones is controlled by `serial-query-rate`.

### 6.2.16.16 Built-in server information zones

The server provides some helpful diagnostic information through a number of built-in zones under the pseudo-top-level-domain `bind` in the CHAOS class. These zones are part of a built-in view (see Section 6.2.23) of class CHAOS which is separate from the default view of class IN. Most global configuration options (allow-query, etc) will apply to this view, but some are locally overridden: notify and recursion are always set to no.

If you need to disable these zones, use the options below, or hide the built-in CHAOS view by defining an explicit view of class CHAOS that matches all clients.

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

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

6.2.16.17  Built-in Empty Zones

Named has some built-in empty zones (SOA and NS records only). These are for zones that should normally be answered locally and which queries should not be sent to the Internet’s root servers. The official servers which cover these namespaces return NXDOMAIN responses to these queries. In particular, these cover the reverse namespaces for addresses from RFC 1918, RFC 4193, RFC 5737 and RFC 6598. They also include the reverse namespace for IPv6 local address (locally assigned), IPv6 link local addresses, the IPv6 loopback address and the IPv6 unknown address.

Named will attempt to determine if a built-in zone already exists or is active (covered by a forward-only forwarding declaration) and will not create a empty zone in that case.

The current list of empty zones is:

- 10.IN-ADDR.ARPA
- 16.172.IN-ADDR.ARPA
- 17.172.IN-ADDR.ARPA
- 18.172.IN-ADDR.ARPA
- 19.172.IN-ADDR.ARPA
- 20.172.IN-ADDR.ARPA
- 21.172.IN-ADDR.ARPA
- 22.172.IN-ADDR.ARPA
- 23.172.IN-ADDR.ARPA
- 24.172.IN-ADDR.ARPA
- 25.172.IN-ADDR.ARPA
- 26.172.IN-ADDR.ARPA
- 27.172.IN-ADDR.ARPA
- 28.172.IN-ADDR.ARPA
- 29.172.IN-ADDR.ARPA
- 30.172.IN-ADDR.ARPA
- 31.172.IN-ADDR.ARPA
- 168.192.IN-ADDR.ARPA
- 64.100.IN-ADDR.ARPA
- 65.100.IN-ADDR.ARPA
- 66.100.IN-ADDR.ARPA
• 67.100.IN-ADDR.ARPA
• 68.100.IN-ADDR.ARPA
• 69.100.IN-ADDR.ARPA
• 70.100.IN-ADDR.ARPA
• 71.100.IN-ADDR.ARPA
• 72.100.IN-ADDR.ARPA
• 73.100.IN-ADDR.ARPA
• 74.100.IN-ADDR.ARPA
• 75.100.IN-ADDR.ARPA
• 76.100.IN-ADDR.ARPA
• 77.100.IN-ADDR.ARPA
• 78.100.IN-ADDR.ARPA
• 79.100.IN-ADDR.ARPA
• 80.100.IN-ADDR.ARPA
• 81.100.IN-ADDR.ARPA
• 82.100.IN-ADDR.ARPA
• 83.100.IN-ADDR.ARPA
• 84.100.IN-ADDR.ARPA
• 85.100.IN-ADDR.ARPA
• 86.100.IN-ADDR.ARPA
• 87.100.IN-ADDR.ARPA
• 88.100.IN-ADDR.ARPA
• 89.100.IN-ADDR.ARPA
• 90.100.IN-ADDR.ARPA
• 91.100.IN-ADDR.ARPA
• 92.100.IN-ADDR.ARPA
• 93.100.IN-ADDR.ARPA
• 94.100.IN-ADDR.ARPA
• 95.100.IN-ADDR.ARPA
• 96.100.IN-ADDR.ARPA
• 97.100.IN-ADDR.ARPA
• 98.100.IN-ADDR.ARPA
• 99.100.IN-ADDR.ARPA
• 100.100.IN-ADDR.ARPA
• 101.100.IN-ADDR.ARPA
• 102.100.IN-ADDR.ARPA
• 103.100.IN-ADDR.ARPA
• 104.100.IN-ADDR.ARPA
• 105.100.IN-ADDR.ARPA
• 106.100.IN-ADDR.ARPA
• 107.100.IN-ADDR.ARPA
• 108.100.IN-ADDR.ARPA
• 109.100.IN-ADDR.ARPA
• 110.100.IN-ADDR.ARPA
• 111.100.IN-ADDR.ARPA
• 112.100.IN-ADDR.ARPA
• 113.100.IN-ADDR.ARPA
• 114.100.IN-ADDR.ARPA
• 115.100.IN-ADDR.ARPA
• 116.100.IN-ADDR.ARPA
• 117.100.IN-ADDR.ARPA
• 118.100.IN-ADDR.ARPA
• 119.100.IN-ADDR.ARPA
• 120.100.IN-ADDR.ARPA
• 121.100.IN-ADDR.ARPA
• 122.100.IN-ADDR.ARPA
• 123.100.IN-ADDR.ARPA
• 124.100.IN-ADDR.ARPA
• 125.100.IN-ADDR.ARPA
• 126.100.IN-ADDR.ARPA
• 127.100.IN-ADDR.ARPA
• 0.IN-ADDR.ARPA
• 127.IN-ADDR.ARPA
• 254.169.IN-ADDR.ARPA
• 2.0.192.IN-ADDR.ARPA
• 100.51.198.IN-ADDR.ARPA
• 113.0.203.IN-ADDR.ARPA
• 255.255.255.255.IN-ADDR.ARPA
• 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA
• 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.IP6.ARPA
• 8.B.D.0.1.0.0.2.IP6.ARPA
• D.F.IP6.ARPA
• 8.E.F.IP6.ARPA
• 9.E.F.IP6.ARPA
• A.E.F.IP6.ARPA
• B.E.F.IP6.ARPA

Empty zones are settable at the view level and only apply to views of class IN. Disabled empty zones are only inherited from options if there are no disabled empty zones specified at the view level. To override the options list of disabled zones, you can disable the root zone at the view level, for example:
disable-empty-zone ".";

If you are using the address ranges covered here, you should already have reverse zones covering the addresses you use. In practice this appears to not be the case with many queries being made to the infrastructure servers for names in these spaces. So many in fact that sacrificial servers were needed to be deployed to channel the query load away from the infrastructure servers.

**Note**

The real parent servers for these zones should disable all empty zone under the parent zone they serve. For the real root servers, this is all built-in empty zones. This will enable them to return referrals to deeper in the tree.

**empty-server** Specify what server name will appear in the returned SOA record for empty zones. If none is specified, then the zone’s name will be used.

**empty-contact** Specify what contact name will appear in the returned SOA record for empty zones. If none is specified, then "." will be used.

**empty-zones-enable** Enable or disable all empty zones. By default, they are enabled.

**disable-empty-zone** Disable individual empty zones. By default, none are disabled. This option can be specified multiple times.

### 6.2.16.18 Additional Section Caching

The additional section cache, also called **acache**, is an internal cache to improve the response performance of BIND 9. When additional section caching is enabled, BIND 9 will cache an internal short-cut to the additional section content for each answer RR. Note that **acache** is an internal caching mechanism of BIND 9, and is not related to the DNS caching server function.

Additional section caching does not change the response content (except the RRsets ordering of the additional section, see below), but can improve the response performance significantly. It is particularly effective when BIND 9 acts as an authoritative server for a zone that has many delegations with many glue RRs.

In order to obtain the maximum performance improvement from additional section caching, setting **additional-from-cache** to **no** is recommended, since the current implementation of **acache** does not short-cut of additional section information from the DNS cache data.

One obvious disadvantage of **acache** is that it requires much more memory for the internal cached data. Thus, if the response performance does not matter and memory consumption is much more critical, the **acache** mechanism can be disabled by setting **acache-enable** to **no**. It is also possible to specify the upper limit of memory consumption for acache by using **max-acache-size**.

Additional section caching also has a minor effect on the RRset ordering in the additional section. Without **acache**, cyclic order is effective for the additional section as well as the answer and authority sections. However, additional section caching fixes the ordering when it first caches an RRset for the additional section, and the same ordering will be kept in succeeding responses, regardless of the setting of **rrset-order**. The effect of this should be minor, however, since an RRset in the additional section typically only contains a small number of RRs (and in many cases it only contains a single RR), in which case the ordering does not matter much.

The following is a summary of options related to **acache**.
acache-enable  If yes, additional section caching is enabled. The default value is no.

acache-cleaning-interval  The server will remove stale cache entries, based on an LRU based algorithm, every acache-cleaning-interval minutes. The default is 60 minutes. If set to 0, no periodic cleaning will occur.

max-acache-size  The maximum amount of memory in bytes to use for the server’s acache. When the amount of data in the acache reaches this limit, the server will clean more aggressively so that the limit is not exceeded. In a server with multiple views, the limit applies separately to the acache of each view. The default is 16M.

6.2.17 server Statement Grammar

server ip_addr[/prefixlen] {
    [ bogus yes_or_no ; ]
    [ provide-ixfr yes_or_no ; ]
    [ request-ixfr yes_or_no ; ]
    [ edns yes_or_no ; ]
    [ edns-udp-size number ; ]
    [ max-udp-size number ; ]
    [ transfers number ; ]
    [ transfer-format (one-answer | many-answers) ; ]
    [ keys { string ; [ string ; [ ... ]] } ; ]
    [ transfer-source (ip4_addr | *) [port ip_port] ; ]
    [ transfer-source-v6 (ip6_addr | *) [port ip_port] ; ]
    [ notify-source (ip4_addr | *) [port ip_port] ; ]
    [ notify-source-v6 (ip6_addr | *) [port ip_port] ; ]
    [ query-source [ address (ip_addr | *) [port (ip_port | *)] ] ; ]
    [ query-source-v6 [ address (ip_addr | *) [port (ip_port | *)] ] ; ]
    [ use-queryport-pool yes_or_no; ]
    [ queryport-pool-ports number; ]
    [ queryport-pool-updateinterval number; ]
};

6.2.18 server Statement Definition and Usage

The server statement defines characteristics to be associated with a remote name server. If a prefix length is specified, then a range of servers is covered. Only the most specific server clause applies regardless of the order in named.conf.

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

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

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

---

67
The `request-ixfr` clause determines whether the local server, acting as a slave, will request incremental zone transfers from the given remote server, a master. If not set, the value of the `request-ixfr` option in the view or global options block is used as a default.

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

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

The `edns-udp-size` option sets the EDNS UDP size that is advertised by `named` when querying the remote server. Valid values are 512 to 4096 bytes (values outside this range will be silently adjusted). This option is useful when you wish to advertise a different value to this server than the value you advertise globally, for example, when there is a firewall at the remote site that is blocking large replies.

The `max-udp-size` option sets the maximum EDNS UDP message size `named` will send. Valid values are 512 to 4096 bytes (values outside this range will be silently adjusted). This option is useful when you know that there is a firewall that is blocking large replies from `named`.

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

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

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

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

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

The `notify-source` and `notify-source-v6` clauses specify the IPv4 and IPv6 source address to be used for notify messages sent to remote servers, respectively. For an IPv4 remote server, only `notify-source` can be specified. Similarly, for an IPv6 remote server, only `notify-source-v6` can be specified.

The `query-source` and `query-source-v6` clauses specify the IPv4 and IPv6 source address to be used for queries sent to remote servers, respectively. For an IPv4 remote server, only `query-source` can be specified. Similarly, for an IPv6 remote server, only `query-source-v6` can be specified.

### 6.2.19 statistics-channels Statement Grammar

```plaintext
statistics-channels {  
    [ inet ( ip_addr | * ) [ port ip_port ] [allow { address_match_list } ]; ]  
    [ inet ...; ]  
};
```
6.2.20 statistics-channels Statement Definition and Usage

The statistics-channels statement declares communication channels to be used by system administrators to get access to statistics information of the name server.

This statement intends to be flexible to support multiple communication protocols in the future, but currently only HTTP access is supported. It requires that BIND 9 be compiled with libxml2; the statistics-channels statement is still accepted even if it is built without the library, but any HTTP access will fail with an error.

An inet control channel is a TCP socket listening at the specified ip_port on the specified ip_addr, which can be an IPv4 or IPv6 address. An ip_addr of * (asterisk) is interpreted as the IPv4 wildcard address; connections will be accepted on any of the system’s IPv4 addresses. To listen on the IPv6 wildcard address, use an ip_addr of ::.

If no port is specified, port 80 is used for HTTP channels. The asterisk "*" cannot be used for ip_port.

The attempt of opening a statistics channel is restricted by the optional allow clause. Connections to the statistics channel are permitted based on the address_match_list. If no allow clause is present, named accepts connection attempts from any address; since the statistics may contain sensitive internal information, it is highly recommended to restrict the source of connection requests appropriately.

If no statistics-channels statement is present, named will not open any communication channels.

6.2.21 trusted-keys Statement Grammar

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

6.2.22 trusted-keys Statement Definition and Usage

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

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

The trusted-keys statement can contain multiple key entries, each consisting of the key’s domain name, flags, protocol, algorithm, and the Base-64 representation of the key data. Spaces, tabs, newlines and carriage returns are ignored in the key data, so the configuration may be split up into multiple lines.

6.2.23 view Statement Grammar

view view_name {
  [class] {
    match-clients { address_match_list };
    match-destinations { address_match_list };
    match-recursive-only yes_or_no ;
    [ view_option; ...]
    [ zone_statement; ...]
  };
}
6.2.24 view Statement Definition and Usage

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

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

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

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

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

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

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

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

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

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

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

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

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

zone zone_name [class] {
    type master;
    [ allow-query { address_match_list }; ]
    [ allow-query-on { address_match_list }; ]
    [ allow-transfer { address_match_list }; ]
    [ allow-update { address_match_list }; ]
    [ update-policy { update_policy_rule [...] }; ]
    [ also-notify { ip_addr [port ip_port] ; [ ip_addr [port ip_port] ; ... ] }; ]
    [ check-names { warn|fail|ignore } ; ]
    [ check-mx { warn|fail|ignore } ; ]
    [ check-wildcard yes_or_no ; ]
    [ check-spf { warn|fail|ignore } ; ]
    [ check-integrity yes_or_no ; ]
    [ dialup dialup_option ; ]
    [ file string ; ]
    [ masterfile-format { text|raw } ; ]
    [ journal string ; ]
    [ max-journal-size size_spec ; ]
    [ forward { only|first } ; ]
    [ forwarders { [ ip_addr [port ip_port] ; ... ] } ; ]
    [ ixfr-base string ; ]
    [ ixfr-from-differences yes_or_no ; ]
    [ ixfr-tmp-file string ; ]
    [ maintain-ixfr-base yes_or_no ; ]
    [ max-ixfr-log-size number ; ]
    [ max-transfer-idle-out number ; ]
    [ max-transfer-time-out number ; ]
    [ notify yes_or_no | explicit | master-only ; ]
    [ notify-delay seconds ; ]
    [ notify-to-soa yes_or_no ; ]
    [ pubkey number number number string ; ]
    [ notify-source { ip4_addr | * } [port ip_port] ; ]
    [ notify-source-v6 { ip6_addr | * } [port ip_port] ; ]
    [ zone-statistics yes_or_no ; ]
    [ sig-validity-interval number [number] ; ]
    [ sig-signing-nodes number ; ]
    [ sig-signing-signatures number ; ]
    [ sig-signing-type number ; ]
    [ database string ; ]
    [ min-refresh-time number ; ]
    [ max-refresh-time number ; ]
    [ min-retry-time number ; ]
    [ max-retry-time number ; ]
    [ key-directory path_name ; ]
    [ zero-no-soa-ttl yes_or_no ; ]
};

zone zone_name [class] {
    type slave;
    [ allow-notify { address_match_list }; ]
    [ allow-query { address_match_list }; ]
    [ allow-query-on { address_match_list }; ]
}
[ allow-transfer { address_match_list }; ]
[ allow-update-forwarding { address_match_list }; ]
[ update-check-ksk yes_or_no; ]
[ try-tcp-refresh yes_or_no; ]
[ also-notify { ip_addr [port ip_port] ; [ ip_addr [port ip_port] ; ... ] } ]; ]
[ check-names (warn|fail|ignore) ; ]
[ dialup dialup_option ; ]
[ file string ; ]
[ masterfile-format (text|raw) ; ]
[ journal string ; ]
[ max-journal-size size_spec; ]
[ forward (only|first) ; ]
[ forwarders { [ ip_addr [port ip_port] ; ... ] } ]; ]
[ ixfr-base string ; ]
[ ixfr-from-differences yes_or_no; ]
[ ixfr-tmp-file string ; ]
[ maintain-ixfr-base yes_or_no ; ]
[ masters [port ip_port] { ( masters_list | ip_addr [port ip_port] [key key] ) ; [ ... ]
[ max-ixfr-log-size number ; ]
[ max-transfer-idle-in number ; ]
[ max-transfer-idle-out number ; ]
[ max-transfer-time-in number ; ]
[ max-transfer-time-out number ; ]
[ notify yes_or_no | explicit | master-only ; ]
[ notify-delay seconds ; ]
[ notify-to-soa yes_or_no; ]
[ pubkey number number number string ; ]
[ transfer-source (ip4_addr | *) [port ip_port] ; ]
[ transfer-source-v6 (ip6_addr | *) [port ip_port] ; ]
[ alt-transfer-source (ip4_addr | *) [port ip_port] ; ]
[ alt-transfer-source-v6 (ip6_addr | *) [port ip_port] ; ]
[ use-alt-transfer-source yes_or_no; ]
[ notify-source (ip4_addr | *) [port ip_port] ; ]
[ notify-source-v6 (ip6_addr | *) [port ip_port] ; ]
[ zone-statistics yes_or_no ; ]
[ database string ; ]
[ min-refresh-time number ; ]
[ max-refresh-time number ; ]
[ min-retry-time number ; ]
[ max-retry-time number ; ]
[ multi-master yes_or_no ; ]
[ zero-no-soa-ttl yes_or_no ; ]
]};

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

zone zone_name [class] {
  type stub;
  [ allow-query { address_match_list }; ]
  [ allow-query-on { address_match_list } ]; ]
  [ check-names (warn|fail|ignore) ; ]
  [ dialup dialup_option ; ]
  [ delegation-only yes_or_no ; ]
  [ file string ; ]
6.2.26 zone Statement Definition and Usage

6.2.26.1 Zone Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>The server has a master copy of the data for the zone and will be able to provide authoritative answers for it.</td>
</tr>
<tr>
<td>slave</td>
<td>A slave zone is a replica of a master zone. The masters list specifies one or more IP addresses of master servers that the slave contacts to update its copy of the zone. Masters list elements can also be names of other masters lists. By default, transfers are made from port 53 on the servers; this can be changed for all servers by specifying a port number before the list of IP addresses, or on a per-server basis after the IP address. Authentication to the master can also be done with per-server TSIG keys. If a file is specified, then the replica will be written to this file whenever the zone is changed, and reloaded from this file on a server restart. Use of a file is recommended, since it often speeds server startup and eliminates a needless waste of bandwidth. Note that for large numbers (in the tens or hundreds of thousands) of zones per server, it is best to use a two-level naming scheme for zone filenames. 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 100000 files into a single directory.)</td>
</tr>
</tbody>
</table>
A stub zone is similar to a slave zone, except that it replicates only the NS records of a master zone instead of the entire zone. Stub zones are not a standard part of the DNS; they are a feature specific to the BIND implementation.

Stub zones can be used to eliminate the need for glue NS record in a parent zone at the expense of maintaining a stub zone entry and a set of name server addresses in named.conf. This usage is not recommended for new configurations, and BIND 9 supports it only in a limited way. In BIND 4/8, zone transfers of a parent zone included the NS records from stub children of that zone. This meant that, in some cases, users could get away with configuring child stubs only in the master server for the parent zone. BIND 9 never mixes together zone data from different zones in this way. Therefore, if a BIND 9 master serving a parent zone has child stub zones configured, all the slave servers for the parent zone also need to have the same child stub zones configured.

Stub zones can also be used as a way of forcing the resolution of a given domain to use a particular set of authoritative servers. For example, the caching name servers on a private network using RFC1918 addressing may be configured with stub zones for 10.in-addr.arpa to use a set of internal name servers as the authoritative servers for that domain.

A “forward zone” is a way to configure forwarding on a per-domain basis. A zone statement of type forward can contain a forward and/or forwarders statement, which will apply to queries within the domain given by the zone name. If no forwarders statement is present or an empty list for forwarders is given, then no forwarding will be done for the domain, canceling the effects of any forwarders in the options statement. Thus if you want to use this type of zone to change the behavior of the global forward option (that is, “forward first” to, then “forward only”, or vice versa, but want to use the same servers as set globally) you need to re-specify the global forwarders.

The initial set of root name servers is specified using a “hint zone”. When the server starts up, it uses the root hints to find a root name server and get the most recent list of root name servers. If no hint zone is specified for class IN, the server uses a compiled-in default set of root servers hints. Classes other than IN have no built-in defaults hints.

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

See caveats in root-delegation-only.

6.2.26.2 Class

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

The hesiod class is named for an information service from MIT’s Project Athena. It is used to share information about various systems databases, such as users, groups, printers and so on. The keyword HS is a synonym for hesiod.
Another MIT development is Chaosnet, a LAN protocol created in the mid-1970s. Zone data for it can be specified with the CHAOS class.

6.2.26.3 Zone Options

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

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

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

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

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


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

also-notify Only meaningful if notify is active for this zone. The set of machines that will receive a DNS NOTIFY message for this zone is made up of all the listed name servers (other than the primary master) for the zone plus any IP addresses specified with also-notify. A port may be specified with each also-notify address to send the notify messages to a port other than the default of 53. also-notify is not meaningful for stub zones. The default is the empty list.

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

check-mx See the description of check-mx in Section 6.2.16.1.

check-spf See the description of check-spf in Section 6.2.16.1.

check-wildcard See the description of check-wildcard in Section 6.2.16.1.

check-integrity See the description of check-integrity in Section 6.2.16.1.

check-sibling See the description of check-sibling in Section 6.2.16.1.

zero-no-soa-ttl See the description of zero-no-soa-ttl in Section 6.2.16.1.

update-check-ksk See the description of update-check-ksk in Section 6.2.16.1.

try-tcp-refresh See the description of try-tcp-refresh in Section 6.2.16.1.

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

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

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

dialup See the description of dialup in Section 6.2.16.1.

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

See caveats in root-delegation-only.

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

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

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


journal Allow the default journal's filename to be overridden. The default is the zone's filename with ".jnl" appended. This is applicable to master and slave zones.

max-journal-size See the description of max-journal-size in Section 6.2.16.10.

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

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

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

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

notify See the description of notify in Section 6.2.16.1.

notify-delay See the description of notify-delay in Section 6.2.16.15.

notify-to-soa See the description of notify-to-soa in Section 6.2.16.1.

pubkey In BIND 8, this option was intended for specifying a public zone key for verification of signatures in DNSSEC signed zones when they are loaded from disk. BIND 9 does not verify signatures on load and ignores the option.
zone-statistics If yes, the server will keep statistical information for this zone, which can be dumped to the statistics-file defined in the server options.

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

sig-signing-nodes See the description of sig-signing-nodes in Section 6.2.16.15.

sig-signing-signatures See the description of sig-signing-signatures in Section 6.2.16.15.

sig-signing-type See the description of sig-signing-type in Section 6.2.16.15.

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

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

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

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

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

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

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

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

ixfr-from-differences See the description of ixfr-from-differences in Section 6.2.16.1. (Note that the ixfr-from-differences master and slave choices are not available at the zone level.)

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

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

masterfile-format See the description of masterfile-format in Section 6.2.16.15.

6.2.26.4 Dynamic Update Policies

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

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

The update-policy clause is new in BIND 9 and allows more fine-grained control over what updates are allowed. A set of rules is specified, where each rule either grants or denies permissions for one or more names to be updated by one or more identities. If the dynamic update request message is signed (that is, it includes either a TSIG or SIG(0) record), the identity of the signer can be determined.
Rules are specified in the `update-policy` zone option, and are only meaningful for master zones. When the `update-policy` statement is present, it is a configuration error for the `allow-update` statement to be present. The `update-policy` statement only examines the signer of a message; the source address is not relevant.

This is how a rule definition looks:

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

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

No signer is required for `tcp-self` or `6to4-self` however the standard reverse mapping / prefix conversion must match the identity field.

The identity field specifies a name or a wildcard name. Normally, this is the name of the TSIG or SIG(0) key used to sign the update request. When a TKEY exchange has been used to create a shared secret, the identity of the shared secret is the same as the identity of the key used to authenticate the TKEY exchange. TKEY is also the negotiation method used by GSS-TSIG, which establishes an identity that is the Kerberos principal of the client, such as `user@host.domain`. When the `identity` field specifies a wildcard name, it is subject to DNS wildcard expansion, so the rule will apply to multiple identities. The `identity` field must contain a fully-qualified domain name.

For nametypes `krb5-self`, `ms-self`, `krb5-subdomain`, and `ms-subdomain` the `identity` field specifies the Windows or Kerberos realm of the machine belongs to.

The `nametype` field has 12 values: `name`, `subdomain`, `wildcard`, `self`, `selfsub`, `selfwild`, `krb5-self`, `ms-self`, `krb5-subdomain`, `ms-subdomain`, `tcp-self` and `6to4-self`.

<table>
<thead>
<tr>
<th>nametype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Exact-match semantics. This rule matches when the name being updated is identical to the contents of the <code>name</code> field.</td>
</tr>
<tr>
<td>subdomain</td>
<td>This rule matches when the name being updated is a subdomain of, or identical to, the contents of the <code>name</code> field.</td>
</tr>
<tr>
<td>wildcard</td>
<td>The <code>name</code> field is subject to DNS wildcard expansion, and this rule matches when the name being updated name is a valid expansion of the wildcard.</td>
</tr>
<tr>
<td>self</td>
<td>This rule matches when the name being updated matches the contents of the <code>identity</code> field. The <code>name</code> field is ignored, but should be the same as the <code>identity</code> field. The <code>self</code> nametype is most useful when allowing one key per name to update, where the key has the same name as the name to be updated. The <code>identity</code> would be specified as <code>*</code> (an asterisk) in this case.</td>
</tr>
<tr>
<td>selfsub</td>
<td>This rule is similar to <code>self</code> except that subdomains of <code>self</code> can also be updated.</td>
</tr>
<tr>
<td>selfwild</td>
<td>This rule is similar to <code>self</code> except that only subdomains of <code>self</code> can be updated.</td>
</tr>
<tr>
<td>ms-self</td>
<td>This rule takes a Windows machine principal (machine$@REALM) for machine in REALM and converts it machine.realm allowing the machine to update machine.realm. The REALM to be matched is specified in the <code>identity</code> field.</td>
</tr>
<tr>
<td>ms-subdomain</td>
<td>This rule takes a Windows machine principal (machine$@REALM) for machine in REALM and converts it to machine.realm allowing the machine to update subdomains of machine.realm. The REALM to be matched is specified in the <code>identity</code> field.</td>
</tr>
</tbody>
</table>
krb5-self  This rule takes a Kerberos machine principal (host/machine@REALM) for machine in REALM and converts it machine.realm allowing the machine to update machine.realm. The REALM to be matched is specified in the identity field.

krb5-subdomain This rule takes a Kerberos machine principal (host/machine@REALM) for machine in REALM and converts it to machine.realm allowing the machine to update subdomains of machine.realm. The REALM to be matched is specified in the identity field.

tcp-self  Allow updates that have been sent via TCP and for which the standard mapping from the initiating IP address into the IN-ADDR.ARPA and IP6.ARPA namespaces match the name to be updated.

**NOTE**  It is theoretically possible to spoof these TCP sessions.

6to4-self Allow the 6to4 prefix to be update by any TCP conection from the 6to4 network or from the corresponding IPv4 address. This is intended to allow NS or DNAME RRsets to be added to the reverse tree.

**NOTE**  It is theoretically possible to spoof these TCP sessions.

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

If no types are explicitly specified, this rule matches all types except RRSIG, NS, SOA, NSEC and NSEC3. Types may be specified by name, including “ANY” (ANY matches all types except NSEC and NSEC3, which can never be updated). Note that when an attempt is made to delete all records associated with a name, the rules are checked for each existing record type.

### 6.3 Zone File

#### 6.3.1 Types of Resource Records and When to Use Them

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

#### 6.3.1.1 Resource Records

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

The components of a Resource Record are:

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

The following are *types* of valid RRs:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A host address. In the IN class, this is a 32-bit IP address. Described in RFC 1035.</td>
</tr>
<tr>
<td>AAAA</td>
<td>IPv6 address. Described in RFC 1886.</td>
</tr>
<tr>
<td>A6</td>
<td>IPv6 address. This can be a partial address (a suffix) and an indirection to the name where the rest of the address (the prefix) can be found. Experimental. Described in RFC 2874.</td>
</tr>
<tr>
<td>APL</td>
<td>Address prefix list. Experimental. Described in RFC 3123.</td>
</tr>
<tr>
<td>CERT</td>
<td>Holds a digital certificate. Described in RFC 2538.</td>
</tr>
<tr>
<td>CNAME</td>
<td>Identifies the canonical name of an alias. Described in RFC 1035.</td>
</tr>
<tr>
<td>DHCID</td>
<td>Is used for identifying which DHCP client is associated with this name. Described in RFC 4701.</td>
</tr>
<tr>
<td>DNAME</td>
<td>Replaces the domain name specified with another name to be looked up, effectively aliasing an entire subtree of the domain name space rather than a single record as in the case of the CNAME RR. Described in RFC 2672.</td>
</tr>
<tr>
<td>DNSKEY</td>
<td>Stores a public key associated with a signed DNS zone. Described in RFC 4034.</td>
</tr>
<tr>
<td>DS</td>
<td>Stores the hash of a public key associated with a signed DNS zone. Described in RFC 4034.</td>
</tr>
<tr>
<td>GPOS</td>
<td>Specifies the global position. Superseded by LOC.</td>
</tr>
<tr>
<td>HINFO</td>
<td>Identifies the CPU and OS used by a host. Described in RFC 1035.</td>
</tr>
<tr>
<td>IPSECKEY</td>
<td>Provides a method for storing IPsec keying material in DNS. Described in RFC 4025.</td>
</tr>
<tr>
<td>KEY</td>
<td>Stores a public key associated with a DNS name. Used in original DNSSEC; replaced by DNSKEY in DNSSECbis, but still used with SIG(0). Described in RFCs 2535 and 2931.</td>
</tr>
<tr>
<td>KX</td>
<td>Identifies a key exchanger for this DNS name. Described in RFC 2230.</td>
</tr>
<tr>
<td>LOC</td>
<td>For storing GPS info. Described in RFC 1876. Experimental.</td>
</tr>
<tr>
<td>MX</td>
<td>Identifies a mail exchange for the domain with a 16-bit preference value (lower is better) followed by the host name of the mail exchange. Described in RFC 974, RFC 1035.</td>
</tr>
</tbody>
</table>
### NAPTR
Name authority pointer. Described in RFC 2915.

### NSAP
A network service access point. Described in RFC 1706.

### NS
The authoritative name server for the domain. Described in RFC 1035.

### NSEC
Used in DNSSECbis to securely indicate that RRs with an owner name in a certain name interval do not exist in a zone and indicate what RR types are present for an existing name. Described in RFC 4034.

### NSEC3
Used in DNSSECbis 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. NSEC3 differs from NSEC in that it prevents zone enumeration but is more computationally expensive on both the server and the client than NSEC. Described in RFC 5155.

### NSEC3PARAM
 Used in DNSSECbis to tell the authoritative server which NSEC3 chains are available to use. Described in RFC 5155.

### 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. Used in original DNSSEC; replaced by NSEC in DNSSECbis. Described in RFC 2535.

### PTR
A pointer to another part of the domain name space. Described in RFC 1035.

### PX
Provides mappings between RFC 822 and X.400 addresses. Described in RFC 2163.

### RP
Information on persons responsible for the domain. Experimental. Described in RFC 1183.

### RRSIG
Contains DNSSECbis signature data. Described in RFC 4034.

### RT
Route-through binding for hosts that do not have their own direct wide area network addresses. Experimental. Described in RFC 1035.

### SIG
Contains DNSSEC signature data. Used in original DNSSEC; replaced by RRSIG in DNSSECbis, but still used for SIG(0). Described in RFCs 2535 and 2931.

### SOA
Identifies the start of a zone of authority. Described in RFC 1035.

### SPF
Contains the Sender Policy Framework information for a given email domain. Described in RFC 4408.

### SRV
Information about well known network services (replaces WKS). Described in RFC 2782.

### SSHFP
Provides a way to securely publish a secure shell key’s fingerprint. Described in RFC 4255.

### TXT
Text records. Described in RFC 1035.

### WKS
Information about which well known network services, such as SMTP, that a domain supports. Historical.

### X25

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

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>The Internet.</td>
</tr>
<tr>
<td>CH</td>
<td>Chaosnet, a LAN protocol created at MIT in the mid-1970s. Rarely used for its historical purpose, but reused for BIND’s built-in server information zones, e.g., version.bind.</td>
</tr>
</tbody>
</table>
The owner name is often implicit, rather than forming an integral part of the RR. For example, many name servers internally form tree or hash structures for the name space, and chain RRs off nodes. The remaining RR parts are the fixed header (type, class, TTL) which is consistent for all RRs, and a variable part (RDATA) that fits the needs of the resource being described.

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

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

### 6.3.1.2 Textual expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In the examples provided in RFC 1034, a style similar to that used in master files was employed in order to show the contents of RRs. In this format, most RRs are shown on a single line, although continuation lines are possible using parentheses. The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

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

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

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

```
<table>
<thead>
<tr>
<th>OWNER</th>
<th>MX</th>
<th>10</th>
<th>VENERA.ISI.EDU.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MX</td>
<td>10</td>
<td>VAXA.ISI.EDU.</td>
</tr>
<tr>
<td>VENERA.ISI.EDU</td>
<td>A</td>
<td>128.9.0.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>10.1.0.52</td>
<td></td>
</tr>
<tr>
<td>VAXA.ISI.EDU</td>
<td>A</td>
<td>10.2.0.27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>128.9.0.33</td>
<td></td>
</tr>
</tbody>
</table>
```

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

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

Similarly we might see:

```
<table>
<thead>
<tr>
<th>OWNER</th>
<th>IN A</th>
<th>10.0.0.44</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH A</td>
<td>MIT.EDU. 2420</td>
</tr>
</tbody>
</table>
```
This example shows two addresses for \texttt{XX.LCS.MIT.EDU}, each of a different class.

### 6.3.2 Discussion of MX Records

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

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

```
<table>
<thead>
<tr>
<th>Domain</th>
<th>Type</th>
<th>Class</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>example.com</td>
<td>MX</td>
<td>A</td>
<td>mail.example.com</td>
</tr>
<tr>
<td>example.com</td>
<td>MX</td>
<td>A</td>
<td>mail2.example.com</td>
</tr>
<tr>
<td>mail.example.com</td>
<td>A</td>
<td></td>
<td>10.0.0.1</td>
</tr>
<tr>
<td>mail2.example.com</td>
<td>A</td>
<td></td>
<td>10.0.0.2</td>
</tr>
</tbody>
</table>
```

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

### 6.3.3 Setting TTLs

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA</td>
<td>The last field in the SOA is the negative caching TTL. This controls how long other servers will cache no-such-domain (NXDOMAIN) responses from you. The maximum time for negative caching is 3 hours (3h).</td>
</tr>
<tr>
<td>$TTL</td>
<td>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.</td>
</tr>
<tr>
<td>RR TTLs</td>
<td>Each RR can have a TTL as the second field in the RR, which will control how long other servers can cache the it.</td>
</tr>
</tbody>
</table>

All of these TTLs default to units of seconds, though units can be explicitly specified, for example, 1h30m.
6.3.4 Inverse Mapping in IPv4

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

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

NOTE

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

6.3.5 Other Zone File Directives

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

6.3.5.1 The @ (at-sign)

When used in the label (or name) field, the asperand or at-sign (@) symbol represents the current origin. At the start of the zone file, it is the `<zone-name>` (followed by trailing dot).

6.3.5.2 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>`, (followed by trailing dot). The current $ORIGIN is appended to the domain specified in the $ORIGIN argument if it is not absolute.

$ORIGIN example.com.
WWW CNAME MAIN-SERVER

is equivalent to

WWW.EXAMPLE.COM. CNAME MAIN-SERVER.EXAMPLE.COM.
6.3.5.3 The $INCLUDE Directive

Syntax: $INCLUDE filename[origin][comment]

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

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

**NOTE**

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

6.3.5.4 The $TTL Directive

Syntax: $TTL default-ttl[comment]

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

$TTL is defined in RFC 2308.

6.3.6 BIND Master File Extension: the $GENERATE Directive

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

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

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

is equivalent to

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

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

For compatibility with earlier versions, $$ is still recognized as indicating a literal $ in the output.

ttl

Specifies the time-to-live of the generated records. If not specified this will be inherited using the normal TTL inheritance rules.

class

Specifies the class of the generated records. This must match the zone class if it is specified.

type

At present the only supported types are PTR, CNAME, DNAME, A, AAAA and NS.

rhs

rhs is a domain name. It is processed similarly to lhs.

The SGENERATE directive is a BIND extension and not part of the standard zone file format. BIND 8 does not support the optional TTL and CLASS fields.

### 6.3.7 Additional File Formats

In addition to the standard textual format, BIND 9 supports the ability to read or dump to zone files in other formats. The raw format is currently available as an additional format. It is a binary format representing BIND 9's internal data structure directly, thereby remarkably improving the loading time.

For a primary server, a zone file in the raw format is expected to be generated from a textual zone file by the named-compilezone command. For a secondary server or for a dynamic zone, it is automatically generated (if this format is specified by the masterfile-format option) when named dumps the zone contents after zone transfer or when applying prior updates.

If a zone file in a binary format needs manual modification, it first must be converted to a textual form by the named-compilezone command. All necessary modification should go to the text file, which should then be converted to the binary form by the named-compilezone command again.

Although the raw format uses the network byte order and avoids architecture-dependent data alignment so that it is as much portable as possible, it is primarily expected to be used inside the same single system. In order to export a zone file in the raw format or make a portable backup of the file, it is recommended to convert the file to the standard textual representation.

### 6.4 BIND9 Statistics

BIND 9 maintains lots of statistics information and provides several interfaces for users to get access to the statistics. The available statistics include all statistics counters that were available in BIND 8 and are meaningful in BIND 9, and other information that is considered useful.

The statistics information is categorized into the following sections.
<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incoming Requests</td>
<td>The number of incoming DNS requests for each OPCODE.</td>
</tr>
<tr>
<td>Incoming Queries</td>
<td>The number of incoming queries for each RR type.</td>
</tr>
<tr>
<td>Outgoing Queries</td>
<td>The number of outgoing queries for each RR type sent from the internal resolver. Maintained per view.</td>
</tr>
<tr>
<td>Name Server Statistics</td>
<td>Statistics counters about incoming request processing.</td>
</tr>
<tr>
<td>Zone Maintenance Statistics</td>
<td>Statistics counters regarding zone maintenance operations such as zone transfers.</td>
</tr>
<tr>
<td>Resolver Statistics</td>
<td>Statistics counters about name resolution performed in the internal resolver. Maintained per view.</td>
</tr>
<tr>
<td>Cache DB RRsets</td>
<td>The number of RRsets per RR type (positive or negative) and nonexistent names stored in the cache database. Maintained per view.</td>
</tr>
<tr>
<td>Socket I/O Statistics</td>
<td>Statistics counters about network related events.</td>
</tr>
</tbody>
</table>

A subset of Name Server Statistics is collected and shown per zone for which the server has the authority when `zone-statistics` is set to `yes`. These statistics counters are shown with their zone and view names. In some cases the view names are omitted for the default view.

There are currently two user interfaces to get access to the statistics. One is in the plain text format dumped to the file specified by the `statistics-file` configuration option. The other is remotely accessible via a statistics channel when the `statistics-channels` statement is specified in the configuration file (see Section 6.2.19.)

### 6.4.1 Statistics Counters

The following tables summarize statistics counters that BIND 9 provides. For each row of the tables, the leftmost column is the abbreviated symbol name of that counter. These symbols are shown in the statistics information accessed via an HTTP statistics channel. The rightmost column gives the description of the counter, which is also shown in the statistics file (but, in this document, possibly with slight
modification for better readability). Additional notes may also be provided in this column. When a middle column exists between these two columns, it gives the corresponding counter name of the BIND 8 statistics, if applicable.

### 6.4.1.1 Name Server Statistics Counters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>BIND8 Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requestv4</td>
<td>RQ</td>
<td>IPv4 requests received. Note: this also counts non query requests.</td>
</tr>
<tr>
<td>Requestv6</td>
<td>RQ</td>
<td>IPv6 requests received. Note: this also counts non query requests.</td>
</tr>
<tr>
<td>ReqEdns0</td>
<td></td>
<td>Requests with EDNS(0) received.</td>
</tr>
<tr>
<td>ReqBadEDNSVer</td>
<td></td>
<td>Requests with unsupported EDNS version received.</td>
</tr>
<tr>
<td>ReqTSIG</td>
<td></td>
<td>Requests with TSIG received.</td>
</tr>
<tr>
<td>ReqSIG0</td>
<td></td>
<td>Requests with SIG(0) received.</td>
</tr>
<tr>
<td>ReqBadSIG</td>
<td></td>
<td>Requests with (TSIG or SIG(0)) signature.</td>
</tr>
<tr>
<td>ReqTCP</td>
<td>RTCP</td>
<td>TCP requests received.</td>
</tr>
<tr>
<td>AuthQryRej</td>
<td>RUQ</td>
<td>Authoritative (non recursive) queries rejected.</td>
</tr>
<tr>
<td>RecQryRej</td>
<td>RURQ</td>
<td>Recursive queries rejected.</td>
</tr>
<tr>
<td>XfrRej</td>
<td>RUXFR</td>
<td>Zone transfer requests rejected.</td>
</tr>
<tr>
<td>UpdateRej</td>
<td>RUUpd</td>
<td>Dynamic update requests rejected.</td>
</tr>
<tr>
<td>Response</td>
<td>SAns</td>
<td>Responses sent.</td>
</tr>
<tr>
<td>RespTruncated</td>
<td></td>
<td>Truncated responses sent.</td>
</tr>
<tr>
<td>RespEDNS0</td>
<td></td>
<td>Responses with EDNS(0) sent.</td>
</tr>
<tr>
<td>RespTSIG</td>
<td></td>
<td>Responses with TSIG sent.</td>
</tr>
<tr>
<td>RespSIG0</td>
<td></td>
<td>Responses with SIG(0) sent.</td>
</tr>
<tr>
<td>QrySuccess</td>
<td></td>
<td>Queries resulted in a successful answer. This means the query which returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a NOERROR response with at least one answer RR. This corresponds to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>success counter of previous versions of BIND 9.</td>
</tr>
<tr>
<td>QryAuthAns</td>
<td></td>
<td>Queries resulted in authoritative answer.</td>
</tr>
<tr>
<td>QryNoauthAns</td>
<td>SNaAns</td>
<td>Queries resulted in non authoritative answer.</td>
</tr>
<tr>
<td>QryReferral</td>
<td></td>
<td>Queries resulted in referral answer. This corresponds to the referral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>counter of previous versions of BIND 9.</td>
</tr>
<tr>
<td>QryNxrrset</td>
<td></td>
<td>Queries resulted in NOERROR responses with no data. This corresponds to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nxrrset counter of previous versions of BIND 9.</td>
</tr>
<tr>
<td>QrySERVFAIL</td>
<td>SFail</td>
<td>Queries resulted in SERVFAIL.</td>
</tr>
<tr>
<td>QryFORMERR</td>
<td>SFErr</td>
<td>Queries resulted in FORMERR.</td>
</tr>
<tr>
<td>QryNXDOMAIN</td>
<td>SNXD</td>
<td>Queries resulted in NODomain.</td>
</tr>
<tr>
<td>QryRecursion</td>
<td>RFwdQ</td>
<td>Queries which caused the server to perform recursion in order to find the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>final answer. This corresponds to the recursion counter of previous versions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of BIND 9.</td>
</tr>
<tr>
<td>QryDuplicate</td>
<td>RDupQ</td>
<td>Queries which the server attempted to recurse but discovered an existing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>query with the same IP address, port, query ID, name, type and class already</td>
</tr>
<tr>
<td></td>
<td></td>
<td>processed. This corresponds to the duplicate counter of previous versions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of BIND 9.</td>
</tr>
<tr>
<td>QryDropped</td>
<td></td>
<td>Recursive queries for which the server discovered an excessive number of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>existing recursive queries for the same name, type and class and were</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subsequently dropped. This is the number of dropped queries due to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reason explained with the clients-per-query and max-clients-per-query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>options (see the description about clients-per-query). This corresponds to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dropped counter of previous versions of BIND 9.</td>
</tr>
</tbody>
</table>
QryFailure

Other query failures. This corresponds to the failure counter of previous versions of BIND 9. Note: this counter is provided mainly for backward compatibility with the previous versions. Normally a more fine-grained counters such as AuthQryRej and RecQryRej that would also fall into this counter are provided, and so this counter would not be of much interest in practice.

XfrReqDone

Requested zone transfers completed.

UpdateReqFwd

Update requests forwarded.

UpdateRespFwd

Update responses forwarded.

UpdateFwdFail

Dynamic update forward failed.

UpdateDone

Dynamic updates completed.

UpdateFail

Dynamic updates failed.

UpdateBadPrereq

Dynamic updates rejected due to prerequisite failure.

6.4.1.2 Zone Maintenance Statistics Counters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotifyOutv4</td>
<td>IPv4 notifies sent.</td>
</tr>
<tr>
<td>NotifyOutv6</td>
<td>IPv6 notifies sent.</td>
</tr>
<tr>
<td>NotifyInv4</td>
<td>IPv4 notifies received.</td>
</tr>
<tr>
<td>NotifyInv6</td>
<td>IPv6 notifies received.</td>
</tr>
<tr>
<td>NotifyRej</td>
<td>Incoming notifies rejected.</td>
</tr>
<tr>
<td>SOAOutv4</td>
<td>IPv4 SOA queries sent.</td>
</tr>
<tr>
<td>SOAOutv6</td>
<td>IPv6 SOA queries sent.</td>
</tr>
<tr>
<td>AXFRReqv4</td>
<td>IPv4 AXFR requested.</td>
</tr>
<tr>
<td>AXFRReqv6</td>
<td>IPv6 AXFR requested.</td>
</tr>
<tr>
<td>IXFRReqv4</td>
<td>IPv4 IXFR requested.</td>
</tr>
<tr>
<td>IXFRReqv6</td>
<td>IPv6 IXFR requested.</td>
</tr>
<tr>
<td>XfrSuccess</td>
<td>Zone transfer requests succeed.</td>
</tr>
<tr>
<td>XfrFail</td>
<td>Zone transfer requests failed.</td>
</tr>
</tbody>
</table>

6.4.1.3 Resolver Statistics Counters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>BIND8 Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queryv4</td>
<td>SFwdQ</td>
<td>IPv4 queries sent.</td>
</tr>
<tr>
<td>Queryv6</td>
<td>SFwdQ</td>
<td>IPv6 queries sent.</td>
</tr>
<tr>
<td>Responsev4</td>
<td>RR</td>
<td>IPv4 responses received.</td>
</tr>
<tr>
<td>Responsev6</td>
<td>RR</td>
<td>IPv6 responses received.</td>
</tr>
<tr>
<td>NXDOMAIN</td>
<td>RNXD</td>
<td>NXDOMAIN received.</td>
</tr>
<tr>
<td>SERVFAIL</td>
<td>RFail</td>
<td>SERVFAIL received.</td>
</tr>
<tr>
<td>FORMERR</td>
<td>RErr</td>
<td>FORMERR received.</td>
</tr>
<tr>
<td>OtherError</td>
<td>RErr</td>
<td>Other errors received.</td>
</tr>
<tr>
<td>EDNS0Fail</td>
<td></td>
<td>EDNS(0) query failures.</td>
</tr>
<tr>
<td>Mismatch</td>
<td>RDupR</td>
<td>Mismatch responses received.</td>
</tr>
<tr>
<td>Truncated</td>
<td></td>
<td>Truncated responses received.</td>
</tr>
<tr>
<td>Lame</td>
<td>RLame</td>
<td>Lame delegations received.</td>
</tr>
<tr>
<td>Retry</td>
<td>SDupQ</td>
<td>Query retries performed.</td>
</tr>
<tr>
<td>QueryAbort</td>
<td></td>
<td>Queries aborted due to quota control.</td>
</tr>
</tbody>
</table>
### QuerySockFail
Failures in opening query sockets. One common reason for such failures is a failure of opening a new socket due to a limitation on file descriptors.

### QueryTimeout
Query timeouts.

### GlueFetchv4 SSysQ
IPv4 NS address fetches invoked.

### GlueFetchv6 SSysQ
IPv6 NS address fetches invoked.

### GlueFetchv4Fail
IPv4 NS address fetch failed.

### GlueFetchv6Fail
IPv6 NS address fetch failed.

### ValAttempt
DNSSEC validation attempted.

### ValOk
DNSSEC validation succeeded.

### ValNegOk
DNSSEC validation on negative information succeeded.

### ValFail
DNSSEC validation failed.

### QryRTTnn
Frequency table on round trip times (RTTs) of queries. Each \( \text{nn} \) specifies the corresponding frequency. In the sequence of \( \text{nn}_1, \text{nn}_2, ..., \text{nn}_m \), the value of \( \text{nn}_i \) is the number of queries whose RTTs are between \( \text{nn}_{(i-1)} \) (inclusive) and \( \text{nn}_i \) (exclusive) milliseconds. For the sake of convenience we define \( \text{nn}_0 \) to be 0. The last entry should be represented as \( \text{nn}_{m+} \), which means the number of queries whose RTTs are equal to or over \( \text{nn}_m \) milliseconds.

---

### 6.4.1.4 Socket I/O Statistics Counters

Socket I/O statistics counters are defined per socket types, which are **UDP4** (UDP/IPv4), **UDP6** (UDP/IPv6), **TCP4** (TCP/IPv4), **TCP6** (TCP/IPv6), **Unix** (Unix Domain), and **FDwatch** (sockets opened outside the socket module). In the following table \(<\text{TYPE}>\) represents a socket type. Not all counters are available for all socket types; exceptions are noted in the description field.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;TYPE&gt;Open</td>
<td>Sockets opened successfully. This counter is not applicable to the <strong>FDwatch</strong> type.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;OpenFail</td>
<td>Failures of opening sockets. This counter is not applicable to the <strong>FDwatch</strong> type.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;Close</td>
<td>Sockets closed.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;BindFail</td>
<td>Failures of binding sockets.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;ConnFail</td>
<td>Failures of connecting sockets.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;Conn</td>
<td>Connections established successfully.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;AcceptFail</td>
<td>Failures of accepting incoming connection requests. This counter is not applicable to the <strong>UDP</strong> and <strong>FDwatch</strong> types.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;Accept</td>
<td>Incoming connections successfully accepted. This counter is not applicable to the <strong>UDP</strong> and <strong>FDwatch</strong> types.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;SendErr</td>
<td>Errors in socket send operations. This counter corresponds to <strong>SErr</strong> counter of <strong>BIND</strong> 8.</td>
</tr>
<tr>
<td>&lt;TYPE&gt;RecvErr</td>
<td>Errors in socket receive operations. This includes errors of send operations on a connected UDP socket notified by an ICMP error message.</td>
</tr>
</tbody>
</table>
6.4.1.5 Compatibility with BIND 8 Counters

Most statistics counters that were available in BIND 8 are also supported in BIND 9 as shown in the above tables. Here are notes about other counters that do not appear in these tables.

**RFwdR,SFwdR** These counters are not supported because BIND 9 does not adopt the notion of *forwarding* as BIND 8 did.

**RAXFR** This counter is accessible in the Incoming Queries section.

**RIQ** This counter is accessible in the Incoming Requests section.

**ROpts** This counter is not supported because BIND 9 does not care about IP options in the first place.
Chapter 7

BIND 9 Security Considerations

7.1 Access Control Lists

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

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

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

Here is an example of how to properly apply ACLs:

```plaintext
// Set up an ACL named "bogusnets" that will block RFC1918 space
// and some reserved space, which is commonly used in spoofing attacks.
acl bogusnets {
    0.0.0.0/8; 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.
7.2 Chroot and Setuid

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

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

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

/usr/local/sbin/named -u 202 -t /var/named

7.2.1 The chroot Environment

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

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

7.2.2 Using the setuid Function

Prior to running the named daemon, use the touch utility (to change file access and modification times) or the chown utility (to set the user id and/or group id) on files to which you want BIND to write.

NOTE

Note that if the named daemon is running as an unprivileged user, it will not be able to bind to new restricted ports if the server is reloaded.

7.3 Dynamic Update Security

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

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

Some sites choose to keep all dynamically-updated DNS data in a subdomain and delegate that subdomain to a separate zone. This way, the top-level zone containing critical data such as the IP addresses of public web and mail servers need not allow dynamic update at all.
Chapter 8

Troubleshooting

8.1 Common Problems

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

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

8.2 Incrementing and Changing the Serial Number

Zone serial numbers are just numbers — they aren’t date related. A lot of people set them to a number that represents a date, usually of the form YYYYMMDDRR. Occasionally they will make a mistake and set them to a “date in the future” then try to correct them by setting them to the “current date”. This causes problems because serial numbers are used to indicate that a zone has been updated. If the serial number on the slave server is lower than the serial number on the master, the slave server will attempt to update its copy of the zone.

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

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

8.3 Where Can I Get Help?

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

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

Appendices

A.1 Acknowledgments

A.1.1 A Brief History of the DNS and BIND

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

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

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

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

In 1994, 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 Systems Consortium and its predecessor, 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 version 9 was released in September 2000 and is a major rewrite of nearly all aspects of the under-lying BIND architecture.

BIND versions 4 and 8 are officially deprecated. No additional development is done on BIND version 4 or BIND version 8.

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

A.2 General DNS Reference Information

A.2.1 IPv6 addresses (AAAA)

IPv6 addresses are 128-bit identifiers for interfaces and sets of interfaces which were introduced in the DNS to facilitate scalable Internet routing. There are three types of addresses: Unicast, an identifier for a single interface; Anycast, an identifier for a set of interfaces; and Multicast, an identifier for a set of interfaces. Here we describe the global Unicast address scheme. For more information, see RFC 3587, “Global Unicast Address Format.”

IPv6 unicast addresses consist of a global routing prefix, a subnet identifier, and an interface identifier.

The global routing prefix is provided by the upstream provider or ISP, and (roughly) corresponds to the IPv4 network section of the address range. The subnet identifier is for local subnetting, much the same as subnetting an IPv4 /16 network into /24 subnets. The interface identifier is the address of an individual interface on a given network; in IPv6, addresses belong to interfaces rather than to machines.

The subnetting capability of IPv6 is much more flexible than that of IPv4: subnetting can be carried out on bit boundaries, in much the same way as Classless InterDomain Routing (CIDR), and the DNS PTR representation (“nibble” format) makes setting up reverse zones easier.

The Interface Identifier must be unique on the local link, and is usually generated automatically by the IPv6 implementation, although it is usually possible to override the default setting if necessary. A typical IPv6 address might look like: 2001:db8:201:9:a00:20ff:fe81:2b32

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

A.3 Bibliography (and Suggested Reading)

A.3.1 Request for Comments (RFCs)

Specification documents for the Internet protocol suite, including the DNS, are published as part of the Request for Comments (RFCs) series of technical notes. The standards themselves are defined by the Internet Engineering Task Force (IETF) and the Internet Engineering Steering Group (IESG). RFCs can be obtained online via FTP at:


(where xxxx is the number of the RFC). RFCs are also available via the Web at:


References
Standards


Proposed Standards

[RFC2931] DNS Request and Transaction Signatures (SIG(0)s), D. Eastlake, 3rd, September 2000.

DNS Security Proposed Standards


Other Important RFCs About DNS Implementation

Resource Record Types


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


[RFC2537]  RSA/MD5 KEYs and SIGs in the Domain Name System (DNS), D. Eastlake, 3rd, March 1999.


DNS and the Internet


DNS Operations


Internationalized Domain Names


Other DNS-related RFCs


Obsolete and Unimplemented Experimental RFC

Obsoleted DNS Security RFCs


A.3.2 Internet Drafts

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

A.3.3 Other Documents About BIND

References

Appendix B

Manual pages

B.1 dig

Name

dig — DNS lookup utility

Synopsis

dig [@server] [-b address] [-c class] [-f filename] [-k filename] [-m] [-p port#] [-q name] [-t type] [-x addr] [-y [hmac:]name:key] [-4] [-6] [name] [type] [class] [queryopt...]  
dig [-h]  
dig [global-queryopt...] [query...]

DESCRIPTION

dig (domain information groper) is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers that are returned from the name server(s) that were queried. Most DNS administrators use dig to troubleshoot DNS problems because of its flexibility, ease of use and clarity of output. Other lookup tools tend to have less functionality than dig.

Although dig is normally used with command-line arguments, it also has a batch mode of operation for reading lookup requests from a file. A brief summary of its command-line arguments and options is printed when the -h option is given. Unlike earlier versions, the BIND 9 implementation of dig allows multiple lookups to be issued from the command line.

Unless it is told to query a specific name server, dig will try each of the servers listed in /etc/resolv.conf. If no usable server addresses are found, dig will send the query to the local host.

When no command line arguments or options are given, dig will perform an NS query for "." (the root).

It is possible to set per-user defaults for dig via $HOME/.digrc. This file is read and any options in it are applied before the command line arguments.

The IN and CH class names overlap with the IN and CH top level domains names. Either use the -t and -c options to specify the type and class, use the -q the specify the domain name, or use "IN." and "CH." when looking up these top level domains.
SIMPLE USAGE

A typical invocation of dig looks like:

dig @server name type

where:

server is the name or IP address of the name server to query. This can be an IPv4 address in dotted-decimal notation or an IPv6 address in colon-delimited notation. When the supplied server argument is a hostname, dig resolves that name before querying that name server.

If no server argument is provided, dig consults /etc/resolv.conf; if an address is found there, it queries the name server at that address. If either of the -4 or -6 options are in use, then only addresses for the corresponding transport will be tried. If no usable addresses are found, dig will send the query to the local host. The reply from the name server that responds is displayed.

name is the name of the resource record that is to be looked up.

type indicates what type of query is required — ANY, A, MX, SIG, etc. type can be any valid query type. If no type argument is supplied, dig will perform a lookup for an A record.

OPTIONS

The -b option sets the source IP address of the query to address. This must be a valid address on one of the host’s network interfaces or "0.0.0.0" or "::". An optional port may be specified by appending ":<port>"

The default query class (IN for internet) is overridden by the -c option. class is any valid class, such as HS for Hesiod records or CH for Chaosnet records.

The -f option makes dig operate in batch mode by reading a list of lookup requests to process from the file filename. The file contains a number of queries, one per line. Each entry in the file should be organized in the same way they would be presented as queries to dig using the command-line interface.

The -m option enables memory usage debugging.

If a non-standard port number is to be queried, the -p option is used. port# is the port number that dig will send its queries instead of the standard DNS port number 53. This option would be used to test a name server that has been configured to listen for queries on a non-standard port number.

The -4 option forces dig to only use IPv4 query transport. The -6 option forces dig to only use IPv6 query transport.

The -t option sets the query type to type. It can be any valid query type which is supported in BIND 9. The default query type is “A”, unless the -x option is supplied to indicate a reverse lookup. A zone transfer can be requested by specifying a type of AXFR. When an incremental zone transfer (IXFR) is required, type is set to ixfr=N. The incremental zone transfer will contain the changes made to the zone since the serial number in the zone’s SOA record was N.

The -q option sets the query name to name. This useful do distinguish the name from other arguments.

Reverse lookups — mapping addresses to names — are simplified by the -x option. addr is an IPv4 address in dotted-decimal notation, or a colon-delimited IPv6 address. When this option is used, there is no need to provide the name, class and type arguments. dig automatically performs a lookup for a name like 11.12.13.10.in-addr.arpa and sets the query type and class to PTR and IN respectively. By default, IPv6 addresses are looked up using nibble format under the IP6.ARPA domain. To use the older RFC1886 method using the IP6.INT domain specify the -i option. Bit string labels (RFC2874) are now experimental and are not attempted.

To sign the DNS queries sent by dig and their responses using transaction signatures (TSIG), specify a TSIG key file using the -k option. You can also specify the TSIG key itself on the command line using
the -y option; hmac is the type of the TSIG, default HMAC-MD5, name is the name of the TSIG key and key is the actual key. The key is a base-64 encoded string, typically generated by dnssec-keygen(8). Caution should be taken when using the -y option on multi-user systems as the key can be visible in the output from ps(1) or in the shell’s history file. When using TSIG authentication with dig, the name server that is queried needs to know the key and algorithm that is being used. In BIND, this is done by providing appropriate key and server statements in named.conf.

QUERY OPTIONS

dig provides a number of query options which affect the way in which lookups are made and the results displayed. Some of these set or reset flag bits in the query header, some determine which sections of the answer get printed, and others determine the timeout and retry strategies.

Each query option is identified by a keyword preceded by a plus sign (+). Some keywords set or reset an option. These may be preceded by the string no to negate the meaning of that keyword. Other keywords assign values to options like the timeout interval. They have the form +keyword=value. The query options are:

+tcp Use TCP when querying name servers. The default behavior is to use UDP unless an AXFR or IXFR query is requested, in which case a TCP connection is used.

+vc Use TCP when querying name servers. This alternate syntax to +tcp is provided for backwards compatibility. The "vc" stands for "virtual circuit".

+ignore Ignore truncation in UDP responses instead of retrying with TCP. By default, TCP retries are performed.

+domain=somename Set the search list to contain the single domain somename, as if specified in a domain directive in /etc/resolv.conf, and enable search list processing as if the +search option were given.

+search Use the search list defined by the searchlist or domain directive in resolv.conf (if any). The search list is not used by default.

+showsearch Perform a search showing intermediate results.

+defname Deprecated, treated as a synonym for +search

+aonly Sets the “aa” flag in the query.

+aflag A synonym for +aonly.

+adflag Set the AD (authentic data) bit in the query. This requests the server to return whether all of the answer and authority sections have all been validated as secure according to the security policy of the server. AD=1 indicates that all records have been validated as secure and the answer is not from a OPT-OUT range. AD=0 indicate that some part of the answer was insecure or not validated.

+cdflag Set the CD (checking disabled) bit in the query. This requests the server to not perform DNSSEC validation of responses.

+cl Display the CLASS when printing the record.
B.1. DIG

APPENDIX B. MANUAL PAGES

+[no]ttlid Display [do not display] the TTL when printing the record.

+[no]recurse Toggle the setting of the RD (recursion desired) bit in the query. This bit is set by default, which means dig normally sends recursive queries. Recursion is automatically disabled when the +nssearch or +trace query options are used.

+[no]nssearch When this option is set, dig attempts to find the authoritative name servers for the zone containing the name being looked up and display the SOA record that each name server has for the zone.

+[no]trace Toggle tracing of the delegation path from the root name servers for the name being looked up. Tracing is disabled by default. When tracing is enabled, dig makes iterative queries to resolve the name being looked up. It will follow referrals from the root servers, showing the answer from each server that was used to resolve the lookup.

+[no]cmd Toggles the printing of the initial comment in the output identifying the version of dig and the query options that have been applied. This comment is printed by default.

+[no]short Provide a terse answer. The default is to print the answer in a verbose form.

+[no]identify Show [or do not show] the IP address and port number that supplied the answer when the +short option is enabled. If short form answers are requested, the default is not to show the source address and port number of the server that provided the answer.

+[no]comments Toggle the display of comment lines in the output. The default is to print comments.

+[no]stats This query option toggles the printing of statistics: when the query was made, the size of the reply and so on. The default behavior is to print the query statistics.

+[no]qr Print [do not print] the query as it is sent. By default, the query is not printed.

+[no]question Print [do not print] the question section of a query when an answer is returned. The default is to print the question section as a comment.

+[no]answer Display [do not display] the answer section of a reply. The default is to display it.

+[no]authority Display [do not display] the authority section of a reply. The default is to display it.

+[no]additional Display [do not display] the additional section of a reply. The default is to display it.

+[no]all Set or clear all display flags.

+time=T Sets the timeout for a query to $T$ seconds. The default timeout is 5 seconds. An attempt to set $T$ to less than 1 will result in a query timeout of 1 second being applied.

+tries=T Sets the number of times to try UDP queries to server to $T$ instead of the default, 3. If $T$ is less than or equal to zero, the number of tries is silently rounded up to 1.
+retry=T Sets the number of times to retry UDP queries to server to \( T \) instead of the default, 2. Unlike +tries, this does not include the initial query.

+ndots=D Set the number of dots that have to appear in name to \( D \) for it to be considered absolute. The default value is that defined using the ndots statement in /etc/resolv.conf, or 1 if no ndots statement is present. Names with fewer dots are interpreted as relative names and will be searched for in the domains listed in the search or domain directive in /etc/resolv.conf.

+bufsize=B Set the UDP message buffer size advertised using EDNS0 to \( B \) bytes. The maximum and minimum sizes of this buffer are 65535 and 0 respectively. Values outside this range are rounded up or down appropriately. Values other than zero will cause a EDNS query to be sent.

+edns=# Specify the EDNS version to query with. Valid values are 0 to 255. Setting the EDNS version will cause a EDNS query to be sent. +noedns clears the remembered EDNS version.

+[no]multiline Print records like the SOA records in a verbose multi-line format with human-readable comments. The default is to print each record on a single line, to facilitate machine parsing of the dig output.

+[no]fail Do not try the next server if you receive a SERVFAIL. The default is to not try the next server which is the reverse of normal stub resolver behavior.

+[no]besteffort Attempt to display the contents of messages which are malformed. The default is to not display malformed answers.

+[no]dnssec Requests DNSSEC records be sent by setting the DNSSEC OK bit (DO) in the OPT record in the additional section of the query.

+[no]sigchase Chase DNSSEC signature chains. Requires dig be compiled with -DDIG_SIGCHASE.

+trusted-key=#### Specifies a file containing trusted keys to be used with +sigchase. Each DNSKEY record must be on its own line.

If not specified, dig will look for /etc/trusted-key.key then trusted-key.key in the current directory.

Requires dig be compiled with -DDIG_SIGCHASE.

+[no]topdown When chasing DNSSEC signature chains perform a top-down validation. Requires dig be compiled with -DDIG_SIGCHASE.

+[no]nsid Include an EDNS name server ID request when sending a query.

+[no]keepopen Keep the TCP socket open between queries and reuse it rather than creating a new TCP socket for each lookup. The default is +nokeepopen.

**MUTLIPLE QUERIES**

The BIND 9 implementation of dig supports specifying multiple queries on the command line (in addition to supporting the -f batch file option). Each of those queries can be supplied with its own set of flags, options and query options.
In this case, each query argument represents an individual query in the command-line syntax described above. Each consists of any of the standard options and flags, the name to be looked up, an optional query type and class and any query options that should be applied to that query.

A global set of query options, which should be applied to all queries, can also be supplied. These global query options must precede the first tuple of name, class, type, options, flags, and query options supplied on the command line. Any global query options (except the +[no]cmd option) can be overridden by a query-specific set of query options. For example:

```
dig +qr www.isc.org any -x 127.0.0.1 isc.org ns +noqr
```

shows how dig could be used from the command line to make three lookups: an ANY query for www.isc.org, a reverse lookup of 127.0.0.1 and a query for the NS records of isc.org. A global query option of +qr is applied, so that dig shows the initial query it made for each lookup. The final query has a local query option of +noqr which means that dig will not print the initial query when it looks up the NS records for isc.org.

IDN SUPPORT

If dig has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. dig appropriately converts character encoding of domain name before sending a request to DNS server or displaying a reply from the server. If you’d like to turn off the IDN support for some reason, defines the IDN_DISABLE environment variable. The IDN support is disabled if the variable is set when dig runs.

FILES

/etc/resolv.conf

${HOME}/.digrc

SEE ALSO

host(1), named(8), dnssec-keygen(8), RFC1035.

BUGS

There are probably too many query options.

B.2 host

Name

host — DNS lookup utility

Synopsis

```
```
**DESCRIPTION**

**host** is a simple utility for performing DNS lookups. It is normally used to convert names to IP addresses and vice versa. When no arguments or options are given, **host** prints a short summary of its command line arguments and options.

**name** is the domain name that is to be looked up. It can also be a dotted-decimal IPv4 address or a colon-delimited IPv6 address, in which case **host** will by default perform a reverse lookup for that address. **server** is an optional argument which is either the name or IP address of the name server that **host** should query instead of the server or servers listed in `/etc/resolv.conf`.

The `-a` (all) option is equivalent to setting the `-v` option and asking **host** to make a query of type ANY.

When the `-c` option is used, **host** will attempt to display the SOA records for zone **name** from all the listed authoritative name servers for that zone. The list of name servers is defined by the NS records that are found for the zone.

The `-c` option instructs to make a DNS query of class **class**. This can be used to lookup Hesiod or Chaosnet class resource records. The default class is IN (Internet).

Verbose output is generated by **host** when the `-d` or `-v` option is used. The two options are equivalent. They have been provided for backwards compatibility. In previous versions, the `-d` option switched on debugging traces and `-v` enabled verbose output.

List mode is selected by the `-l` option. This makes **host** perform a zone transfer for zone **name**. Transfer the zone printing out the NS, PTR and address records (A/AAAA). If combined with `-a` all records will be printed.

The `-i` option specifies that reverse lookups of IPv6 addresses should use the IP6.INT domain as defined in RFC1886. The default is to use IP6.ARPA.

The `-n` option sets the number of dots that have to be in **name** for it to be considered absolute. The default value is that defined using the ndots statement in `/etc/resolv.conf`, or 1 if no ndots statement is present. Names with fewer dots are interpreted as relative names and will be searched for in the domains listed in the search or domain directive in `/etc/resolv.conf`.

The number of UDP retries for a lookup can be changed with the `-R` option. `number` indicates how many times **host** will repeat a query that does not get answered. The default number of retries is 1. If `number` is negative or zero, the number of retries will default to 1.

Non-recursive queries can be made via the `-r` option. Setting this option clears the RD — recursion desired — bit in the query which **host** makes. This should mean that the name server receiving the query will not attempt to resolve **name**. The `-r` option enables **host** to mimic the behavior of a name server by making non-recursive queries and expecting to receive answers to those queries that are usually referrals to other name servers.

By default, **host** uses UDP when making queries. The `-T` option makes it use a TCP connection when querying the name server. TCP will be automatically selected for queries that require it, such as zone transfer (AXFR) requests.

The `-4` option forces **host** to only use IPv4 query transport. The `-6` option forces **host** to only use IPv6 query transport.

The `-t` option is used to select the query type. **type** can be any recognized query type: CNAME, NS, SOA, SIG, KEY, AXFR, etc. When no query type is specified, **host** automatically selects an appropriate query type. By default, it looks for A, AAAA, and MX records, but if the `-C` option was given, queries will be made for SOA records, and if **name** is a dotted-decimal IPv4 address or colon-delimited IPv6 address, **host** will query for PTR records. If a query type of IXFR is chosen the starting serial number can be specified by appending an equal followed by the starting serial number (e.g. `-t IXFR=12345678`).

The time to wait for a reply can be controlled through the `-W` and `-w` options. The `-W` option makes **host** wait for `wait` seconds. If `wait` is less than one, the wait interval is set to one second. When the `-w` option is used, **host** will effectively wait forever for a reply. The time to wait for a response will be set to the number of seconds given by the hardware's maximum value for an integer quantity.
The `-s` option tells `host` not to send the query to the next nameserver if any server responds with a SERVFAIL response, which is the reverse of normal stub resolver behavior.

The `-m` can be used to set the memory usage debugging flags `record`, `usage` and `trace`.

**IDN SUPPORT**

If `host` has been built with IDN (internationalized domain name) support, it can accept and display non-ASCII domain names. `host` appropriately converts character encoding of domain name before sending a request to DNS server or displaying a reply from the server. If you’d like to turn off the IDN support for some reason, defines the `IDN_DISABLE` environment variable. The IDN support is disabled if the variable is set when `host` runs.

**FILES**

```
/etc/resolv.conf
```

**SEE ALSO**

dig(1), named(8).

---

**B.3 dnssec-dsfromkey**

**Name**

dnssec-dsfromkey — DNSSEC DS RR generation tool

**Synopsis**

```
```

```
```

**DESCRIPTION**

dnssec-dsfromkey outputs the Delegation Signer (DS) resource record (RR), as defined in RFC 3658 and RFC 4509, for the given key(s).

**OPTIONS**

- `-1` Use SHA-1 as the digest algorithm (the default is to use both SHA-1 and SHA-256).

- `-2` Use SHA-256 as the digest algorithm.

- `-a algorithm` Select the digest algorithm. The value of `algorithm` must be one of SHA-1 (SHA1) or SHA-256 (SHA256). These values are case insensitive.
**APPENDIX B. MANUAL PAGES**  

**B.4. DNSSEC-KEYFROMLABEL**

- **v level** Sets the debugging level.

- **s** Keyset mode: in place of the keyfile name, the argument is the DNS domain name of a keyset file. Following options make sense only in this mode.

- **c class** Specifies the DNS class (default is IN), useful only in the keyset mode.

- **d directory** Look for keyset files in directory as the directory, ignored when not in the keyset mode.

**EXAMPLE**

To build the SHA-256 DS RR from the **Kexample.com.+003+26160** keyfile name, the following command would be issued:

```
  dnssec-dsfromkey -2 Kexample.com.+003+26160
```

The command would print something like:

```
  example.com.  IN  DS  26160  5 2 3A1EADA7A74B8D0BA86726B0C227AA85AB8BBD2B2004F41A868A54FC5EA0B94
```

**FILES**

The keyfile can be designed by the key identification **Knnnn.+aaa+iiiii** or the full file name **Knnnn.+aaa+iiiii.key** as generated by dnssec-keygen(8).

The keyset file name is built from the directory, the string **keyset-** and the dnsname.

**CAVEAT**

A keyfile error can give a “file not found” even if the file exists.

**SEE ALSO**


**AUTHOR**

Internet Systems Consortium

**B.4 dnssec-keyfromlabel**

**Name**

dnssec-keyfromlabel — DNSSEC key generation tool
Synopsis


DESCRIPTION

dnssec-keyfromlabel gets keys with the given label from a crypto hardware and builds key files for DNSSEC (Secure DNS), as defined in RFC 2535 and RFC 4034.

OPTIONS

-a algorithm Selects the cryptographic algorithm. The value of algorithm must be one of RSAMD5, RSASHA1, DSA, NSEC3RSASHA1, NSEC3DSA, RSASHA256, RSASHA512 or DH (Diffie Hellman). These values are case insensitive.

If no algorithm is specified, then RSASHA1 will be used by default, unless the -3 option is specified, in which case NSEC3RSASHA1 will be used instead. (If -3 is used and an algorithm is specified, that algorithm will be checked for compatibility with NSEC3.)

Note 1: that for DNSSEC, RSASHA1 is a mandatory to implement algorithm, and DSA is recommended.

Note 2: DH automatically sets the -k flag.

-l label Specifies the label of keys in the crypto hardware (PKCS#11 device).

-n nametype Specifies the owner type of the key. The value of nametype must either be ZONE (for a DNSSEC zone key (KEY/DNSKEY)), HOST or ENTITY (for a key associated with a host (KEY)), USER (for a key associated with a user(KE)) or OTHER (DNSKEY). These values are case insensitive.

-c class Indicates that the DNS record containing the key should have the specified class. If not specified, class IN is used.

-f flag Set the specified flag in the flag field of the KEY/DNSKEY record. The only recognized flag is KSK (Key Signing Key) DNSKEY.

-h Prints a short summary of the options and arguments to dnssec-keygen.

-k Generate KEY records rather than DNSKEY records.

-p protocol Sets the protocol value for the generated key. The protocol is a number between 0 and 255. The default is 3 (DNSSEC). Other possible values for this argument are listed in RFC 2535 and its successors.

-t type Indicates the use of the key. type must be one of AUTHCONF, NOAUTHCONF, NOAUTH, or NOCONF. The default is AUTHCONF. AUTH refers to the ability to authenticate data, and CONF the ability to encrypt data.

-v level Sets the debugging level.
**GENERATED KEY FILES**

When `dnssec-keyfromlabel` completes successfully, it prints a string of the form `Knnnn.+aaa+iiiii` to the standard output. This is an identification string for the key files it has generated.

- **nnnn** is the key name.
- **aaa** is the numeric representation of the algorithm.
- **iiiii** is the key identifier (or footprint).

`dnssec-keyfromlabel` creates two files, with names based on the printed string, `Knnnn.+aaa+iiiii.key` contains the public key, and `Knnnn.+aaa+iiiii.private` contains the private key.

The `.key` file contains a DNS KEY record that can be inserted into a zone file (directly or with a `$INCLUDE` statement).

The `.private` file contains algorithm specific fields. For obvious security reasons, this file does not have general read permission.

**SEE ALSO**

`dnssec-keygen(8), dnssec-signzone(8), BIND 9 Administrator Reference Manual, RFC 4034`

**AUTHOR**

Internet Systems Consortium

**B.5 dnssec-keygen**

**Name**

dnssec-keygen — DNSSEC key generation tool

**Synopsis**

```bash
```

**DESCRIPTION**

dnssec-keygen generates keys for DNSSEC (Secure DNS), as defined in RFC 2535 and RFC 4034. It can also generate keys for use with TSIG (Transaction Signatures), as defined in RFC 2845.

The name of the key is specified on the command line. For DNSSEC keys, this must match the name of the zone for which the key is being generated.

**OPTIONS**

- `-a algorithm` Selects the cryptographic algorithm. For DNSSEC keys, the value of `algorithm` must be one of RSAMD5, RSASHA1, DSA, NSEC3RSASHA1, NSEC3DSA, RSASHA256 or RSASHA512.
For TSIG/TKEY, the value must be DH (Diffie Hellman), HMAC-MD5, HMAC-SHA1, HMAC-SHA224, HMAC-SHA256, HMAC-SHA384, or HMAC-SHA512. These values are case insensitive.

Note 1: that for DNSSEC, RSASHA1 is a mandatory to implement algorithm, and DSA is recommended. For TSIG, HMAC-MD5 is mandatory.

Note 2: HMAC-MD5 and DH automatically set the -k flag.

-b **keysize** Specifies the number of bits in the key. The choice of key size depends on the algorithm used. RSA keys must be between 512 and 2048 bits. Diffie Hellman keys must be between 128 and 4096 bits. DSA keys must be between 512 and 1024 bits and an exact multiple of 64. HMAC keys must be between 1 and 512 bits.

-n **nametype** Specifies the owner type of the key. The value of nametype must either be ZONE (for a DNSSEC zone key (KEY/DNSKEY)), HOST or ENTITY (for a key associated with a host (KEY)), USER (for a key associated with a user(KEY)) or OTHER (DNSKEY). These values are case insensitive. Defaults to ZONE for DNSKEY generation.

-c **class** Indicates that the DNS record containing the key should have the specified class. If not specified, class IN is used.

-e If generating an RSAMD5/RSASHA1 key, use a large exponent.

-f **flag** Set the specified flag in the flag field of the KEY/DNSKEY record. The only recognized flag is KSK (Key Signing Key) DNSKEY.

-g **generator** If generating a Diffie Hellman key, use this generator. Allowed values are 2 and 5. If no generator is specified, a known prime from RFC 2539 will be used if possible; otherwise the default is 2.

-h Prints a short summary of the options and arguments to **dnssec-keygen**.

-k Generate KEY records rather than DNSKEY records.

-p **protocol** Sets the protocol value for the generated key. The protocol is a number between 0 and 255. The default is 3 (DNSSEC). Other possible values for this argument are listed in RFC 2535 and its successors.

-r **randomdev** Specifies the source of randomness. If the operating system does not provide a /dev/random or equivalent device, the default source of randomness is keyboard input. randomdev specifies the name of a character device or file containing random data to be used instead of the default. The special value keyboard indicates that keyboard input should be used.

-s **strength** Specifies the strength value of the key. The strength is a number between 0 and 15, and currently has no defined purpose in DNSSEC.

-t **type** Indicates the use of the key. type must be one of AUTHCONF, NOAUTHCONF, NOAUTH, or NOCONF. The default is AUTHCONF. AUTH refers to the ability to authenticate data, and CONF the ability to encrypt data.

-v **level** Sets the debugging level.
GENERATED KEYS

When `dnssec-keygen` completes successfully, it prints a string of the form `Knnnn.+aaa+iiiii` to the standard output. This is an identification string for the key it has generated.

- `nnnn` is the key name.
- `aaa` is the numeric representation of the algorithm.
- `iiiii` is the key identifier (or footprint).

`dnssec-keygen` creates two files, with names based on the printed string. `Knnnn.+aaa+iiiii.key` contains the public key, and `Knnnn.+aaa+iiiii.private` contains the private key.

The `.key` file contains a DNS KEY record that can be inserted into a zone file (directly or with a `$INCLUDE` statement).

The `.private` file contains algorithm-specific fields. For obvious security reasons, this file does not have general read permission.

Both `.key` and `.private` files are generated for symmetric encryption algorithms such as HMAC-MD5, even though the public and private key are equivalent.

EXAMPLE

To generate a 768-bit DSA key for the domain `example.com`, the following command would be issued:

```
dnssec-keygen -a DSA -b 768 -n ZONE example.com
```

The command would print a string of the form:

```
Kexample.com.+003+26160
```

In this example, `dnssec-keygen` creates the files `Kexample.com.+003+26160.key` and `Kexample.com.+003+26160.private`.

SEE ALSO


AUTHOR

Internet Systems Consortium

B.6 dnssec-signzone

Name

dnssec-signzone — DNSSEC zone signing tool

Synopsis

```
```
DESCRIPTION

dnsec-signzone signs a zone. It generates NSEC and RRSIG records and produces a signed version of
the zone. It also generates a keyset- file containing the key-signing keys for the zone, and if signing
a zone which contains delegations, it can optionally generate DS records for the child zones from their
keyset- files.

OPTIONS

-a Verify all generated signatures.

-c class Specifies the DNS class of the zone.

-k key Treat specified key as a key signing key ignoring any key flags. This option may be specified
multiple times.

-l domain Generate a DLV set in addition to the key (DNSKEY) and DS sets. The domain is appended
to the name of the records.

-d directory Look for keyset files in directory as the directory

-g If the zone contains any delegations, and there are keyset- files for any of the child zones, then DS
records for the child zones will be generated from the keys in those files. Existing DS records will
be removed.

-s start-time Specify the date and time when the generated RRSIG records become valid. This can be
either an absolute or relative time. An absolute start time is indicated by a number in YYYYMMDD-DHHMMSS notation; 20000530144500 denotes 14:45:00 UTC on May 30th, 2000. A relative start
time is indicated by +N, which is N seconds from the current time. If no start-time is specified,
the current time minus 1 hour (to allow for clock skew) is used.

-e end-time Specify the date and time when the generated RRSIG records expire. As with start-time,
an absolute time is indicated in YYYYMMDDHHMMSS notation. A time relative to the start time
is indicated with +N, which is N seconds from the start time. A time relative to the current time
is indicated with now+N. If no end-time is specified, 30 days from the start time is used as a
default.

-f output-file The name of the output file containing the signed zone. The default is to append .signed to the input filename.

-h Prints a short summary of the options and arguments to dnsec-signzone.

-i interval When a previously-signed zone is passed as input, records may be resigned. The interval
option specifies the cycle interval as an offset from the current time (in seconds). If a RRSIG record
expires after the cycle interval, it is retained. Otherwise, it is considered to be expiring soon, and
it will be replaced.

The default cycle interval is one quarter of the difference between the signature end and start times.
So if neither end-time or start-time are specified, dnsec-signzone generates signatures that
are valid for 30 days, with a cycle interval of 7.5 days. Therefore, if any existing RRSIG records are
due to expire in less than 7.5 days, they would be replaced.
-l input-format  The format of the input zone file. Possible formats are “text” (default) and “raw”. This option is primarily intended to be used for dynamic signed zones so that the dumped zone file in a non-text format containing updates can be signed directly. The use of this option does not make much sense for non-dynamic zones.

-j jitter  When signing a zone with a fixed signature lifetime, all RRSIG records issued at the time of signing expires simultaneously. If the zone is incrementally signed, i.e. a previously-signed zone is passed as input to the signer, all expired signatures have to be regenerated at about the same time. The jitter option specifies a jitter window that will be used to randomize the signature expire time, thus spreading incremental signature regeneration over time.

Signature lifetime jitter also to some extent benefits validators and servers by spreading out cache expiration, i.e. if large numbers of RRSIGs don’t expire at the same time from all caches there will be less congestion than if all validators need to refetch at mostly the same time.

-n ncpus  Specifies the number of threads to use. By default, one thread is started for each detected CPU.

-N soa-serial-format  The SOA serial number format of the signed zone. Possible formats are “keep” (default), “increment” and “unixtime”.

“keep”  Do not modify the SOA serial number.

“increment”  Increment the SOA serial number using RFC 1982 arithmetics.

“unixtime”  Set the SOA serial number to the number of seconds since epoch.

-o origin  The zone origin. If not specified, the name of the zone file is assumed to be the origin.

-O output-format  The format of the output file containing the signed zone. Possible formats are “text” (default) and “raw”.

-p  Use pseudo-random data when signing the zone. This is faster, but less secure, than using real random data. This option may be useful when signing large zones or when the entropy source is limited.

-P  Disable post sign verification tests.

The post sign verification test ensures that for each algorithm in use there is at least one non revoked self signed KSK key, that all revoked KSK keys are self signed, and that all records in the zone are signed by the algorithm. This option skips these tests.

-r randomdev  Specifies the source of randomness. If the operating system does not provide a /dev/random or equivalent device, the default source of randomness is keyboard input. randomdev specifies the name of a character device or file containing random data to be used instead of the default. The special value keyboard indicates that keyboard input should be used.

-t  Print statistics at completion.

-v level  Sets the debugging level.
-z  Ignore KSK flag on key when determining what to sign.

-salt  Generate a NSEC3 chain with the given hex encoded salt. A dash (salt) can be used to indicate that no salt is to be used when generating the NSEC3 chain.

-H  iterations  When generating an NSEC3 chain, use this many iterations. The default is 100.

-A  When generating a NSEC3 chain set the OPTOUT flag on all NSEC3 records and do not generate NSEC3 records for insecure delegations.

zonefile  The file containing the zone to be signed.

key  Specify which keys should be used to sign the zone. If no keys are specified, then the zone will be examined for DNSKEY records at the zone apex. If these are found and there are matching private keys, in the current directory, then these will be used for signing.

EXAMPLE

The following command signs the example.com zone with the DSA key generated by dnssec-keygen (Kexample.com.+003+17247). The zone’s keys must be in the master file (db.example.com). This invocation looks for keyset files, in the current directory, so that DS records can be generated from them (-g).

% dnssec-signzone -g -o example.com db.example.com \Kexample.com.+003+17247
        db.example.com.signed
%

In the above example, dnssec-signzone creates the file db.example.com.signed. This file should be referenced in a zone statement in a named.conf file.

This example re-signs a previously signed zone with default parameters. The private keys are assumed to be in the current directory.

% cp db.example.com.signed db.example.com
% dnssec-signzone -o example.com db.example.com db.example.com.signed
%

KNOWN BUGS

dnssec-signzone was designed so that it could sign a zone partially, using only a subset of the DNSSEC keys needed to produce a fully-signed zone. This permits a zone administrator, for example, to sign a zone with one key on one machine, move the resulting partially-signed zone to a second machine, and sign it again with a second key.

An unfortunate side-effect of this flexibility is that dnssec-signzone does not check to make sure it’s signing a zone with any valid keys at all. An attempt to sign a zone without any keys will appear to succeed, producing a “signed” zone with no signatures. There is no warning issued when a zone is not fully signed.

This will be corrected in a future release. In the meantime, ISC recommends examining the output of dnssec-signzone to confirm that the zone is properly signed by all keys before using it.
SEE ALSO


AUTHOR

Internet Systems Consortium

B.7 named-checkconf

Name
	named-checkconf — named configuration file syntax checking tool

Synopsis

named-checkconf [-h] [-v] [-j] [-t directory] filename [-z]

DESCRIPTION

named-checkconf checks the syntax, but not the semantics, of a named configuration file.

OPTIONS

-h  Print the usage summary and exit.

-t directory  Chroot to directory so that include directives in the configuration file are processed as if run by a similarly chrooted named.

-v  Print the version of the named-checkconf program and exit.

-z  Perform a test load of all master zones found in named.conf.

filename  The name of the configuration file to be checked. If not specified, it defaults to /etc/named.conf.

RETURN VALUES

named-checkconf returns an exit status of 1 if errors were detected and 0 otherwise.

SEE ALSO

named(8), named-checkzone(8), BIND 9 Administrator Reference Manual.
B.8  named-checkzone

NAME
	named-checkzone, named-compilezone — zone file validity checking or converting tool

SYNOPSIS

[-i mode] [-k mode] [-m mode] [-M mode] [-n mode] [-s style] [-S mode]


DESCRIPTION

named-checkzone checks the syntax and integrity of a zone file. It performs the same checks as named
does when loading a zone. This makes named-checkzone useful for checking zone files before config-
uring them into a name server.

named-compilezone is similar to named-checkzone, but it always dumps the zone contents to a spec-
ified file in a specified format. Additionally, it applies stricter check levels by default, since the dump
output will be used as an actual zone file loaded by named. When manually specified otherwise, the
check levels must at least be as strict as those specified in the named configuration file.

OPTIONS

-d  Enable debugging.

-h  Print the usage summary and exit.

-q  Quiet mode - exit code only.

-v  Print the version of the named-checkzone program and exit.

-j  When loading the zone file read the journal if it exists.

-c class  Specify the class of the zone. If not specified, "IN" is assumed.

-i mode  Perform post-load zone integrity checks. Possible modes are "full" (default), "full-sibling",
"local", "local-sibling" and "none".
Mode “full” checks that MX records refer to A or AAAA record (both in-zone and out-of-zone hostnames). Mode “local” only checks MX records which refer to in-zone hostnames.

Mode “full” checks that SRV records refer to A or AAAA record (both in-zone and out-of-zone hostnames). Mode “local” only checks SRV records which refer to in-zone hostnames.

Mode “full” checks that delegation NS records refer to A or AAAA record (both in-zone and out-of-zone hostnames). It also checks that glue address records in the zone match those advertised by the child. Mode “local” only checks NS records which refer to in-zone hostnames or that some required glue exists, that is when the nameserver is in a child zone.

Mode “full-sibling” and “local-sibling” disable sibling glue checks but are otherwise the same as “full” and “local” respectively.

Mode “none” disables the checks.

-f format Specify the format of the zone file. Possible formats are “text” (default) and “raw”.

-F format Specify the format of the output file specified. Possible formats are “text” (default) and “raw”. For named-checkzone, this does not cause any effects unless it dumps the zone contents.

-k mode Perform “check-names” checks with the specified failure mode. Possible modes are “fail” (default for named-compilezone), “warn” (default for named-checkzone) and “ignore”.

-m mode Specify whether MX records should be checked to see if they are addresses. Possible modes are “fail”, “warn” (default) and “ignore”.

-M mode Check if a MX record refers to a CNAME. Possible modes are “fail”, “warn” (default) and “ignore”.

-n mode Specify whether NS records should be checked to see if they are addresses. Possible modes are “fail” (default for named-compilezone), “warn” (default for named-checkzone) and “ignore”.

-o filename Write zone output to filename. If filename is – then write to standard out. This is mandatory for named-compilezone.

-s style Specify the style of the dumped zone file. Possible styles are “full” (default) and “relative”. The full format is most suitable for processing automatically by a separate script. On the other hand, the relative format is more human-readable and is thus suitable for editing by hand. For named-checkzone this does not cause any effects unless it dumps the zone contents. It also does not have any meaning if the output format is not text.

-S mode Check if a SRV record refers to a CNAME. Possible modes are “fail”, “warn” (default) and “ignore”.

-t directory Chroot to directory so that include directives in the configuration file are processed as if run by a similarly chrooted named.

-T mode Check if Sender Policy Framework records (TXT and SPF) both exist or both don’t exist. A warning is issued if they don’t match. Possible modes are “warn” (default), “ignore”.

-w directory chdir to directory so that relative filenames in master file $INCLUDE directives work. This is similar to the directory clause in named.conf.
-D Dump zone file in canonical format. This is always enabled for named-compilezone.

-W mode Specify whether to check for non-terminal wildcards. Non-terminal wildcards are almost always the result of a failure to understand the wildcard matching algorithm (RFC 1034). Possible modes are "warn" (default) and "ignore".

zonename The domain name of the zone being checked.

filename The name of the zone file.

RETURN VALUES

named-checkzone returns an exit status of 1 if errors were detected and 0 otherwise.

SEE ALSO

named(8), named-checkconf(8), RFC 1035, BIND 9 Administrator Reference Manual.

AUTHOR

Internet Systems Consortium

B.9 named

Name

named — Internet domain name server

Synopsis


DESCRIPTION

named is a Domain Name System (DNS) server, part of the BIND 9 distribution from ISC. For more information on the DNS, see RFCs 1033, 1034, and 1035.

When invoked without arguments, named will read the default configuration file /etc/named.conf, read any initial data, and listen for queries.
OPTIONS

-4 Use IPv4 only even if the host machine is capable of IPv6. -4 and -6 are mutually exclusive.

-6 Use IPv6 only even if the host machine is capable of IPv4. -4 and -6 are mutually exclusive.

-c config-file Use config-file as the configuration file instead of the default, /etc/named.conf. To ensure that reloading the configuration file continues to work after the server has changed its working directory due to a possible directory option in the configuration file, config-file should be an absolute pathname.

-d debug-level Set the daemon’s debug level to debug-level. Debugging traces from named become more verbose as the debug level increases.

-f Run the server in the foreground (i.e. do not daemonize).

-g Run the server in the foreground and force all logging to stderr.

-m flag Turn on memory usage debugging flags. Possible flags are usage, trace, record, size, and mctx. These correspond to the ISC_MEM_DEBUGXXXX flags described in <isc/mem.h>.

-n #cpus Create #cpus worker threads to take advantage of multiple CPUs. If not specified, named will try to determine the number of CPUs present and create one thread per CPU. If it is unable to determine the number of CPUs, a single worker thread will be created.

-p port Listen for queries on port port. If not specified, the default is port 53.

-s Write memory usage statistics to stdout on exit.

NOTE

This option is mainly of interest to BIND 9 developers and may be removed or changed in a future release.

-S #max-socks Allow named to use up to #max-socks sockets.

WARNING

This option should be unnecessary for the vast majority of users. The use of this option could even be harmful because the specified value may exceed the limitation of the underlying system API. It is therefore set only when the default configuration causes exhaustion of file descriptors and the operational environment is known to support the specified number of sockets. Note also that the actual maximum number is normally a little fewer than the specified value because named reserves some file descriptors for its internal use.
-t directory  Chroot to directory after processing the command line arguments, but before reading the configuration file.

**WARNING**

This option should be used in conjunction with the -u option, as chrooting a process running as root doesn’t enhance security on most systems; the way chroot(2) is defined allows a process with root privileges to escape a chroot jail.

-u user  Setuid to user after completing privileged operations, such as creating sockets that listen on privileged ports.

**NOTE**

On Linux, named uses the kernel’s capability mechanism to drop all root privileges except the ability to bind(2) to a privileged port and set process resource limits. Unfortunately, this means that the -u option only works when named is run on kernel 2.2.18 or later, or kernel 2.3.99-pre3 or later, since previous kernels did not allow privileges to be retained after setuid(2).

-v  Report the version number and exit.

-V  Report the version number and build options, and exit.

-x cache-file  Load data from cache-file into the cache of the default view.

**WARNING**

This option must not be used. It is only of interest to BIND 9 developers and may be removed or changed in a future release.

**SIGNALS**

In routine operation, signals should not be used to control the nameserver; rndc should be used instead.

SIGHUP  Force a reload of the server.

SIGINT, SIGTERM  Shut down the server.

The result of sending any other signals to the server is undefined.
CONFIGURATION

The named configuration file is too complex to describe in detail here. A complete description is provided in the BIND 9 Administrator Reference Manual.

named inherits the umask (file creation mode mask) from the parent process. If files created by named, such as journal files, need to have custom permissions, the umask should be set explicitly in the script used to start the named process.

FILES

/etc/named.conf The default configuration file.
/var/run/named/named.pid The default process-id file.

SEE ALSO

RFC 1033, RFC 1034, RFC 1035, named-checkconf(8), named-checkzone(8), rndc(8), lwresd(8), named.conf(5), BIND 9 Administrator Reference Manual.

AUTHOR

Internet Systems Consortium

B.10 nsupdate

Name

nsupdate — Dynamic DNS update utility

Synopsis


DESCRIPTION

nsupdate is used to submit Dynamic DNS Update requests as defined in RFC2136 to a name server. This allows resource records to be added or removed from a zone without manually editing the zone file. A single update request can contain requests to add or remove more than one resource record.

Zones that are under dynamic control via nsupdate or a DHCP server should not be edited by hand. Manual edits could conflict with dynamic updates and cause data to be lost.

The resource records that are dynamically added or removed with nsupdate have to be in the same zone. Requests are sent to the zone’s master server. This is identified by the MNAME field of the zone’s SOA record.

The -d option makes nsupdate operate in debug mode. This provides tracing information about the update requests that are made and the replies received from the name server.

The -D option makes nsupdate report additional debugging information to -d.
Transaction signatures can be used to authenticate the Dynamic DNS updates. These use the TSIG resource record type described in RFC2845 or the SIG(0) record described in RFC3535 and RFC2931 or GSS-TSIG as described in RFC3645. TSIG relies on a shared secret that should only be known to \texttt{nsupdate} and the name server. Currently, the only supported encryption algorithm for TSIG is HMAC-MD5, which is defined in RFC 2104. Once other algorithms are defined for TSIG, applications will need to ensure they select the appropriate algorithm as well as the key when authenticating each other. For instance, suitable key and server statements would be added to /etc/named.conf so that the name server can associate the appropriate secret key and algorithm with the IP address of the client application that will be using TSIG authentication. SIG(0) uses public key cryptography. To use a SIG(0) key, the public key must be stored in a KEY record in a zone served by the name server. \texttt{nsupdate} does not read /etc/named.conf. GSS-TSIG uses Kerberos credentials.

\texttt{nsupdate} uses the \texttt{\_y} or \texttt{-k} option to provide the shared secret needed to generate a TSIG record for authenticating Dynamic DNS update requests, default type HMAC-MD5. These options are mutually exclusive. With the \texttt{-k} option, \texttt{nsupdate} reads the shared secret from the file keyfile, whose name is of the form $K\{\text{name}\}.+157.+(\text{random}).private$. For historical reasons, the file $K\{\text{name}\}.+157.+(\text{random}).key$ must also be present. When the \texttt{-y} option is used, a signature is generated from $[\text{hmac}:\text{keyname}:\text{secret}]$. keyname is the name of the key, and secret is the base64 encoded shared secret. Use of the \texttt{-y} option is discouraged because the shared secret is supplied as a command line argument in clear text. This may be visible in the output from ps(1) or in a history file maintained by the user’s shell.

The \texttt{-k} may also be used to specify a SIG(0) key used to authenticate Dynamic DNS update requests. In this case, the key specified is not an HMAC-MD5 key.

The \texttt{-g} and \texttt{-o} specify that GSS-TSIG is to be used. The \texttt{-o} should only be used with old Microsoft Windows 2000 servers.

By default, \texttt{nsupdate} uses UDP to send update requests to the name server unless they are too large to fit in a UDP request in which case TCP will be used. The \texttt{-v} option makes \texttt{nsupdate} use a TCP connection. This may be preferable when a batch of update requests is made.

The \texttt{-t} option sets the maximum time an update request can take before it is aborted. The default is 300 seconds. Zero can be used to disable the timeout.

The \texttt{-u} option sets the UDP retry interval. The default is 3 seconds. If zero, the interval will be computed from the timeout interval and number of UDP retries.

The \texttt{-r} option sets the number of UDP retries. The default is 3. If zero, only one update request will be made.

The \texttt{-R randomdev} option specifies a source of randomness. If the operating system does not provide a /dev/random or equivalent device, the default source of randomness is keyboard input. randomdev specifies the name of a character device or file containing random data to be used instead of the default. The special value keyboard indicates that keyboard input should be used. This option may be specified multiple times.

**INPUT FORMAT**

\texttt{nsupdate} reads input from \texttt{filename} or standard input. Each command is supplied on exactly one line of input. Some commands are for administrative purposes. The others are either update instructions or prerequisite checks on the contents of the zone. These checks set conditions that some name or set of resource records (RRSet) either exists or is absent from the zone. These conditions must be met if the entire update request is to succeed. Updates will be rejected if the tests for the prerequisite conditions fail.

Every update request consists of zero or more prerequisites and zero or more updates. This allows a suitably authenticated update request to proceed if some specified resource records are present or missing from the zone. A blank input line (or the \texttt{send} command) causes the accumulated commands to be sent as one Dynamic DNS update request to the name server.

The command formats and their meaning are as follows:
server servername [port] Sends all dynamic update requests to the name server servername. When no server statement is provided, nsupdate will send updates to the master server of the correct zone. The MNAME field of that zone’s SOA record will identify the master server for that zone. 

port is the port number on servername where the dynamic update requests get sent. If no port number is specified, the default DNS port number of 53 is used.

local address [port] Sends all dynamic update requests using the local address. When no local statement is provided, nsupdate will send updates using an address and port chosen by the system. 

port can additionally be used to make requests come from a specific port. If no port number is specified, the system will assign one.

zone zonename Specifies that all updates are to be made to the zone zonename. If no zone statement is provided, nsupdate will attempt determine the correct zone to update based on the rest of the input.

class classname Specify the default class. If no class is specified, the default class is IN.

ttl seconds Specify the default time to live for records to be added. The value none will clear the default ttl.

key name secret Specifies that all updates are to be TSIG-signed using the keyname keysecret pair. 

The key command overrides any key specified on the command line via −y or −k.

gsstsig Use GSS-TSIG to sign the updated. This is equivalent to specifying −g on the commandline.

oldgsstsig Use the Windows 2000 version of GSS-TSIG to sign the updated. This is equivalent to specifying −o on the commandline.

realm [realm_name] When using GSS-TSIG use realm_name rather than the default realm in krb5.conf. If no realm is specified the saved realm is cleared.

prereq nxdomain domain-name Requires that no resource record of any type exists with name domain-name.

prereq yxdomain domain-name Requires that domain-name exists (has as at least one resource record, of any type).

prereq nxrrset domain-name [class] type Requires that no resource record exists of the specified type, class and domain-name. If class is omitted, IN (internet) is assumed.

prereq yxrrset domain-name [class] type This requires that a resource record of the specified type, class and domain-name must exist. If class is omitted, IN (internet) is assumed.

prereq yxrrset domain-name [class] type data... The data from each set of prerequisites of this form sharing a common type, class, and domain-name are combined to form a set of RRs. This set of RRs must exactly match the set of RRs existing in the zone at the given type, class, and domain-name. The data are written in the standard text representation of the resource record’s RDATA.

update delete domain-name [ttl] [class] [type [data...]] Deletes any resource records named domain-name. 

If type and data is provided, only matching resource records will be removed. The internet class
is assumed if class is not supplied. The ttl is ignored, and is only allowed for compatibility.

**update add domain-name ttl [class] type data...** Adds a new resource record with the specified **ttl**, **class** and **data**.

**show** Displays the current message, containing all of the prerequisites and updates specified since the last send.

**send** Sends the current message. This is equivalent to entering a blank line.

**answer** Displays the answer.

**debug** Turn on debugging.

Lines beginning with a semicolon are comments and are ignored.

**EXAMPLES**

The examples below show how nsupdate could be used to insert and delete resource records from the example.com zone. Notice that the input in each example contains a trailing blank line so that a group of commands are sent as one dynamic update request to the master name server for example.com.

```plaintext
# nsupdate
> update delete oldhost.example.com A
> update add newhost.example.com 86400 A 172.16.1.1
> send
```

Any A records for oldhost.example.com are deleted. And an A record for newhost.example.com with IP address 172.16.1.1 is added. The newly-added record has a 1 day TTL (86400 seconds).

```plaintext
# nsupdate
> prereq nxdomain nickname.example.com
> update add nickname.example.com 86400 CNAME somehost.example.com
> send
```

The prerequisite condition gets the name server to check that there are no resource records of any type for nickname.example.com. If there are, the update request fails. If this name does not exist, a CNAME for it is added. This ensures that when the CNAME is added, it cannot conflict with the long-standing rule in RFC1034 that a name must not exist as any other record type if it exists as a CNAME. (The rule has been updated for DNSSEC in RFC2535 to allow CNAMEs to have RRSIG, DNSKEY and NSEC records.)

**FILES**

/etc/resolv.conf used to identify default name server

```
K{name}.+157.+{random}.key base-64 encoding of HMAC-MD5 key created by dnssec-keygen(8).
K{name}.+157.+{random}.private base-64 encoding of HMAC-MD5 key created by dnssec-keygen(8).
```
SEE ALSO

RFC2136, RFC3007, RFC2104, RFC2845, RFC1034, RFC2535, RFC2931, named(8), dnssec-keygen(8).

BUGS

The TSIG key is redundantly stored in two separate files. This is a consequence of nsupdate using the DST library for its cryptographic operations, and may change in future releases.

B.11  rndc

Name

rndc — name server control utility

Synopsis


DESCRIPTION

rndc controls the operation of a name server. It supersedes the ndc utility that was provided in old BIND releases. If rndc is invoked with no command line options or arguments, it prints a short summary of the supported commands and the available options and their arguments.

rndc communicates with the name server over a TCP connection, sending commands authenticated with digital signatures. In the current versions of rndc and named, the only supported authentication algorithm is HMAC-MD5, which uses a shared secret on each end of the connection. This provides TSIG-style authentication for the command request and the name server’s response. All commands sent over the channel must be signed by a key_id known to the server.

rndc reads a configuration file to determine how to contact the name server and decide what algorithm and key it should use.

OPTIONS

-b source-address Use source-address as the source address for the connection to the server. Multiple instances are permitted to allow setting of both the IPv4 and IPv6 source addresses.

-c config-file Use config-file as the configuration file instead of the default, /etc/rndc.conf.

-k key-file Use key-file as the key file instead of the default, /etc/rndc.key. The key in /etc/rndc.key will be used to authenticate commands sent to the server if the config-file does not exist.

-s server server is the name or address of the server which matches a server statement in the configuration file for rndc. If no server is supplied on the command line, the host named by the default-server clause in the options statement of the rndc configuration file will be used.
-p port  Send commands to TCP port port instead of BIND 9's default control channel port, 953.

-V  Enable verbose logging.

-y key_id  Use the key key_id from the configuration file. key_id must be known by named with the same algorithm and secret string in order for control message validation to succeed. If no key_id is specified, rndc will first look for a key clause in the server statement of the server being used, or if no server statement is present for that host, then the default-key clause of the options statement. Note that the configuration file contains shared secrets which are used to send authenticated control commands to name servers. It should therefore not have general read or write access.

COMMANDS

A list of commands supported by rndc can be seen by running rndc without arguments.

Currently supported commands are:

reload  Reload configuration file and zones.

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

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

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

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

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

notify zone [class [view]]  Resend NOTIFY messages for the zone.

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

stats  Write server statistics to the statistics file.

querylog [on|off]  Toggle query logging. Query logging can also be enabled by explicitly directing the queries category to a channel in the logging section of named.conf or by specifying querylog yes; in the options section of named.conf.

dumpdb [-all|-cache|-zone] [view ...]  Dump the server's caches (default) and/or zones to the dump file for the specified views. If no view is specified, all views are dumped.
**stop** [-p] Stop the server, making sure any recent changes made through dynamic update or IXFR are first saved to the master files of the updated zones. If -p is specified named’s process id is returned. This allows an external process to determine when named had completed stopping.

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

**trace** Increment the servers debugging level by one.

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

**notrace** Sets the server’s debugging level to 0.

**flush** Flushes the server’s cache.

**flushname name [view]** Flashes the given name from the server’s cache.

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

**recurring** Dump the list of queries named is currently recursing on.

**validation ( on | off | check ) [view ...]** Enable, disable, or check the current status of DNSSEC validation. Note dnssec-enable also needs to be set to yes or auto to be effective. It defaults to enabled.

**tsig-list** List the names of all TSIG keys currently configured for use by named in each view. The list both statically configured keys and dynamic TKEY-negotiated keys.

**tsig-delete keyname [view]** Delete a given TKEY-negotiated key from the server. (This does not apply to statically configured TSIG keys.)

**LIMITATIONS**

There is currently no way to provide the shared secret for a key.id without using the configuration file. Several error messages could be clearer.

**SEE ALSO**


**AUTHOR**

Internet Systems Consortium
B.12  rndc.conf

Name

rndc.conf — rndc configuration file

Synopsis

rndc.conf

DESCRIPTION

rndc.conf is the configuration file for rndc, the BIND 9 name server control utility. This file has a similar structure and syntax to named.conf. Statements are enclosed in braces and terminated with a semi-colon.Clauses in the statements are also semi-colon terminated. The usual comment styles are supported:

C style: /* */
C++ style: // to end of line
Unix style: # to end of line

rndc.conf is much simpler than named.conf. The file uses three statements: an options statement, a server statement and a key statement.

The options statement contains five clauses. The default-server clause is followed by the name or address of a name server. This host will be used when no name server is given as an argument to rndc. The default-key clause is followed by the name of a key which is identified by a key statement. If no keyid is provided on the rndc command line, and no key clause is found in a matching server statement, this default key will be used to authenticate the server’s commands and responses. The default-port clause is followed by the port to connect to on the remote name server. If no port option is provided on the rndc command line, and no port clause is found in a matching server statement, this default port will be used to connect. The default-source-address and default-source-address-v6 clauses which can be used to set the IPv4 and IPv6 source addresses respectively.

After the server keyword, the server statement includes a string which is the hostname or address for a name server. The statement has three possible clauses: key, port and addresses. The key name must match the name of a key statement in the file. The port number specifies the port to connect to. If an addresses clause is supplied these addresses will be used instead of the server name. Each address can take an optional port. If an source-address or source-address-v6 of supplied then these will be used to specify the IPv4 and IPv6 source addresses respectively.

The key statement begins with an identifying string, the name of the key. The statement has two clauses. algorithm identifies the encryption algorithm for rndc to use; currently only HMAC-MD5 is supported. This is followed by a secret clause which contains the base-64 encoding of the algorithm’s encryption key. The base-64 string is enclosed in double quotes.

There are two common ways to generate the base-64 string for the secret. The BIND 9 program rndc-confgen can be used to generate a random key, or the mmencode program, also known as mimencode, can be used to generate a base-64 string from known input. mmencode does not ship with BIND 9 but is available on many systems. See the EXAMPLE section for sample command lines for each.

EXAMPLE

    options { 

default-server localhost;
default-key samplekey;
};

server localhost {
  key samplekey;
};

server testserver {
  key testkey;
  addresses { localhost port 5353; }
};

key samplekey {
  algorithm hmac-md5;
  secret "6FMfj43Oszi4lyb240Ie2iGEz9lf11lJ0+1z";
};

key testkey {
  algorithm hmac-md5;
  secret "R3Hl8P6BKw9ZwXwN3hZKuQ==";
};

In the above example, **rndc** will by default use the server at localhost (127.0.0.1) and the key called samplekey. Commands to the localhost server will use the samplekey key, which must also be defined in the server's configuration file with the same name and secret. The key statement indicates that samplekey uses the HMAC-MD5 algorithm and its secret clause contains the base-64 encoding of the HMAC-MD5 secret enclosed in double quotes.

If **rndc -s testserver** is used then **rndc** will connect to server on localhost port 5353 using the key testkey.

To generate a random secret with **rndc-confgen**:

```bash
rndc-confgen
```

A complete **rndc.conf** file, including the randomly generated key, will be written to the standard output. Commented-out key and controls statements for **named.conf** are also printed.

To generate a base-64 secret with **mmencode**:

```bash
echo "known plaintext for a secret" | mmencode
```

**NAME SERVER CONFIGURATION**

The name server must be configured to accept rndc connections and to recognize the key specified in the **rndc.conf** file, using the controls statement in **named.conf**. See the sections on the controls statement in the BIND 9 Administrator Reference Manual for details.

**SEE ALSO**

rndc(8), rndc-confgen(8), mmencode(1), BIND 9 Administrator Reference Manual.
AUTHOR

Internet Systems Consortium

B.13  rndc-confgen

Name

rndc-confgen — rndc key generation tool

Synopsis


DESCRIPTION

rndc-confgen generates configuration files for rndc. It can be used as a convenient alternative to writing the rndc.conf file and the corresponding controls and key statements in named.conf by hand. Alternatively, it can be run with the -a option to set up a rndc.key file and avoid the need for a rndc.conf file and a controls statement altogether.

OPTIONS

-a  Do automatic rndc configuration. This creates a file rndc.key in /etc (or whatever sysconfdir was specified as when BIND was built) that is read by both rndc and named on startup. The rndc.key file defines a default command channel and authentication key allowing rndc to communicate with named on the local host with no further configuration.

Running rndc-confgen -a allows BIND 9 and rndc to be used as drop-in replacements for BIND 8 and ndc, with no changes to the existing BIND 8 named.conf file.

If a more elaborate configuration than that generated by rndc-confgen -a is required, for example if rndc is to be used remotely, you should run rndc-confgen without the -a option and set up a rndc.conf and named.conf as directed.

-b keysize  Specifies the size of the authentication key in bits. Must be between 1 and 512 bits; the default is 128.

-c keyfile  Used with the -a option to specify an alternate location for rndc.key.

-h  Prints a short summary of the options and arguments to rndc-confgen.

-k keyname  Specifies the key name of the rndc authentication key. This must be a valid domain name. The default is rndc-key.

-p port  Specifies the command channel port where named listens for connections from rndc. The default is 953.
-r randomfile Specifies a source of random data for generating the authorization. If the operating system does not provide a /dev/random or equivalent device, the default source of randomness is keyboard input. randomdev specifies the name of a character device or file containing random data to be used instead of the default. The special value keyboard indicates that keyboard input should be used.

-s address Specifies the IP address where named listens for command channel connections from rndc. The default is the loopback address 127.0.0.1.

-t chrootdir Used with the -a option to specify a directory where named will run chrooted. An additional copy of the rndc.key will be written relative to this directory so that it will be found by the chrooted named.

-u user Used with the -a option to set the owner of the rndc.key file generated. If -t is also specified only the file in the chroot area has its owner changed.

EXAMPLES
To allow rndc to be used with no manual configuration, run

```
rndc-confgen -a
```  
To print a sample rndc.conf file and corresponding controls and key statements to be manually inserted into named.conf, run

```
rndc-confgen
```  

SEE ALSO

AUTHOR
Internet Systems Consortium