Name Server Operations Guide for BIND

Release 4.9.2

Releases from 4.9

Paul Vixie[1]

<paul@vix.com>

Vixie Enterprises

Redwood City, CA

Releases through 4.8.3

Kevin J. Dunlap[2]

Michael J. Karels

Computer Systems Research Group

Computer Science Division

Department of Electrical Engineering and Computer Sciences

University of California

Berkeley CA 94720

1. Introduction

The Berkeley Internet Name Domain (BIND) implements an Internet name server for the UNIX† operating system. The BIND consists of a server (or ‘‘daemon’’) and a *resolver* library. A name server is a network service that enables clients to name resources or objects and share this information with other objects in the network. This in effect is a distributed data base system for objects in a computer network. BIND is fully integrated into BSD (4.3 and later releases) network programs for use in storing and retrieving host names and address. The system administrator can configure the system to use BIND as a replacement to the older host table lookup of information in the network hosts file */etc/hosts*. The default configuration for BSD uses BIND.

2. Building A System with a Name Server

BIND is composed of two parts. One is the user interface called the *resolver* which consists of a group of routines that reside in the C library */lib/libc.a*. Second is the actual server called *named*. This is a daemon that runs in the background and services queries on a given network port. The standard port for UDP and TCP is specified in */etc/services*.

---

[1] This author was employed by Digital Equipment Corporation’s Network Systems Laboratory during the development and release of BIND 4.9. Releases from 4.9.2 were sponsored by Vixie Enterprises.

[2] This author was an employee of Digital Equipment Corporation’s Ultrix Engineering Advanced Development Group and was on loan to CSRG when this work was done. Ultrix is a trademark of Digital Equipment Corporation.

2.1. Resolver Routines in libc

When building your 4.3BSD system you may either build the C library to use the name server resolver routines or use the host table lookup routines to do host name and address resolution. The default resolver for 4.3BSD uses the name server. Newer BSD systems include both name server and host table functionality with preference given to the name server if there is one or if there is a */etc/resolv.conf* file.

Building the C library to use the name server changes the way *gethostbyname* (3N), *gethostbyaddr* (3N), and *sethostent* (3N) do their functions. The name server renders *gethostent* (3N) obsolete, since it has no concept of a next line in the database. These library calls are built with the resolver routines needed to query the name server.

The *resolver* contains functions that build query packets and exchange them with name servers.

Before building the 4.3BSD C library, set the variable *HOSTLOOKUP* equal to *named* in */ usr/ src/ lib/ libc/ Makefile*. You then make and install the C library and compiler and then compile the rest of the 4.3BSD system. For more information see section 6.6 of ‘‘Installing and Operating 4.3BSD on the VAX‡’’.

If your operating system isn't VAX‡ 4.3BSD, it is probably the case that your vendor has included *resolver* support in the supplied C Library. You should consult your vendor's documentation to find out what has to be done to enable *resolver* support. Note that your vendor's *resolver* may be out of date with respect to the one shipped with BIND, and that you might want to build BIND's resolver library and install it, and its include files, into your system's compile/link path so that your own network applications will be able to use the newer features.

2.2. The Name Service

The basic function of the name server is to provide information about network objects by answering queries. The specifications for this name server are defined in RFC1034, RFC1035 and RFC974. These documents can be found in */usr/src/etc/named/doc* in 4.3BSD or *ftp*ed from **ftp.rs.internic.net**. It is also recommended that you read the related manual pages, *named* (8), *resolver* (3), and *resolver* (5).

The advantage of using a name server over the host table lookup for host name resolution is to avoid the need for a single centralized clearinghouse for all names. The authority for this information can be delegated to the different organizations on the network responsible for it.

The host table lookup routines require that the master file for the entire network be maintained at a central location by a few people. This works fine for small networks where there are only a few machines and the different organizations responsible for them cooperate. But this does not work well for large networks where machines cross organizational boundaries.

With the name server, the network can be broken into a hierarchy of domains. The name space is organized as a tree according to organizational or administrative boundaries. Each node, called a *domain*, is given a label, and the name of the domain is the concatenation of all the labels of the domains from the root to the current domain, listed from right to left separated by dots. A label need only be unique within its domain. The whole space is partitioned into several areas called *zones*, each starting at a domain and extending down to the leaf domains or to domains where other zones start. Zones usually represent administrative boundaries. An example of a host address for a host at the University of California, Berkeley would look as follows:

monet **.** *Berkeley* **.** *EDU*

The top level domain for educational organizations is EDU; Berkeley is a subdomain of EDU and monet is the name of the host.

‡VAX is a Trademark of Digital Equipment Corporation

2.3. About Hesiod, and HS-class Resource Records

Hesiod, developed by MIT Project Athena, is an information service built upon BIND. Its intent is similar to that of Sun's NIS: to furnish information about users, groups, network-accessible file systems, printcaps, and mail service throughout an installation. Aside from its use of BIND rather than separate server code another important difference between Hesiod and NIS is that Hesiod is not intended to deal with passwords and authentication, but only with data that are not security sensitive. Hesiod servers can be implemented by adding resource records to BIND servers; or they can be implemented as separate servers separately administered.

To learn about and obtain Hesiod make an anonymous FTP connection to host ATHENA-DIST.MIT.EDU and retrieve the compressed tar file **pub/hesiod.tar.Z**. You will not need the named and resolver library portions of the distribution because their functionality has already been integrated into BIND 4.9. To learn how Hesiod functions as part of the Athena computing environment obtain the paper **pub/usenix/athena-changes.PS** from the above FTP server host. There is also a tar file of sample Hesiod resource files.

Whether one should use Hesiod class is open to question, since the same services can probably be provided with class IN, type TXT and type CNAME records. In either case, the code and documents for Hesiod will suggest how to set up and use the service.

Note that while BIND includes support for *HS*-class queries, the zone transfer logic for non-*IN*-class zones is still experimental.

2.4. About ''secure zones''

Secure zones implement named security on a zone by zone basis. It is designed to use a permission list of networks or hosts which may obtain particular information from the zone.

In order to use zone security, named must be compiled with SECURE_ZONES defined and you must have at least one secure_zone TXT RR. Unless a secure_zone record exists for a given zone, no restrictions will be applied to the data in that zone. The format of the secure_zone TXT RR is:

```
secure_zone      addr-class      TXT      string
```

The addr-class may be either HS or IN. The syntax for the TXT string is either "network address:netmask" or "host IP address:H".

"network address:netmask" allows queries from an entire network. If the netmask is omitted, named will use the default netmask for the network address specified.

"host IP address:H" allows queries from a host. The "H" after the ":" is required to differentiate the host address from a network address. Multiple secure_zone TXT RRs are allowed in the same zone file.

For example, you can set up a zone to only answer hesiod requests from the masked class B network 130.215.0.0 and from host 128.23.10.56 by adding the following two TXT RR's:

```
secure_zone      HS      TXT      "130.215.0.0:255.255.0.0"
secure_zone      HS      TXT      "128.23.10.56:H"
```

This feature can be used to restrict access to a Hesiod password map or to seperate internal and external internet address resolution on a firewall machine without needing to run a seperate named for internal and external address resolution.

3. Types of Zones

A ''zone'' is a point of delegation in the DNS tree. It contains all names from a certain point ''downward'' except those which are delegated to other servers. A ''delegation point'' has one or more *NS* records in the ''parent zone'', which should be matched by equivalent *NS* records at the root of the ''delegated zone'' (i.e., the ''@'' name in the zone file).

Understanding the difference between a ‘‘zone’’ and a ‘‘domain’’ is crucial to the proper operation of a name server. As an example, consider the DEC.COM *domain*, which includes names such as POBOX1.PA.DEC.COM and QUABBIN.CRL.DEC.COM even though the DEC.COM *zone* includes only *delegations* for the PA.DEC.COM and CRL.DEC.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). Technically speaking, every name in the DNS tree is a ‘‘domain’’, even if it is ‘‘terminal’’, that is, has no ‘‘subdomains’’. Technically speaking, every subdomain is a domain and every domain except the root is also a subdomain. The terminology is not intuitive and you would do well to read RFC’s 1033, 1034, and 1035 to gain a complete understanding of this difficult and subtle topic.

Though BIND is a *Domain* Name Server, it deals primarily in terms of *zones*. The *primary* and *secondary* declarations in the *named.boot* file specify *zones*, not *domains*. When you ask someone if they are willing to be a secondary server for your ‘‘domain’’, you are actually asking for secondary service for some collection of *zones*.

Each zone will have one ‘‘primary’’ server, which loads the zone contents from some local file which is edited by humans or perhaps generated mechanically from some other local file which is edited by humans. Then there will be some number of ‘‘secondary’’ servers, which load the zone contents using the IP/DNS protocol (that is, the secondary servers will contact the primary and fetch the zone using IP/TCP). This set of servers (the primary and all of the secondaries) should be listed in the *NS* records in the parent zone, which will constitute a ‘‘delegation’’. This set of servers must also be listed in the zone file itself, usually under the ‘‘@’’ name which is a magic cookie that means the ‘‘top level’’ or ‘‘root’’ of current $ORIGIN. You can list servers in the zone’s top-level ‘‘@’’ *NS* records that are not in the parent’s *NS* delegation, but you cannot list servers in the parent’s delegation that are not present in the zone’s ‘‘@’’. (This latter condition is one form of what is called a ‘‘lame delegation’’.)

4. Types of Servers

Servers do not really have ‘‘types’’. A server can be a primary for some zones and a secondary for others, or it can be only a primary, or only a secondary, or it can serve no zones and just answer queries via its ‘‘cache’’. Previous versions of this document referred to servers as ‘‘master’’ and ‘‘slave’’ but we now feel that those distinctions — and the assignment of a ‘‘type’’ to a name server — are not useful.

4.1. Caching Only Server

All servers are caching servers. This means that the server caches the information that it receives for use until the data expires. A *Caching Only Server* is a server that is not authoritative for any domain. This server services queries and asks other servers, who have the authority, for the information needed. All servers keep data in their cache until the data expires, based on a *TTL* (‘‘Time To Live’’) field which is maintained for all resource records.

4.2. Remote Server

A Remote Server is an option given to people who would like to use a name server from their workstation or on a machine that has a limited amount of memory and CPU cycles. With this option you can run all of the networking programs that use the name server without the name server running on the local machine. All of the queries are serviced by a name server that is running on another machine on the network. This kind of host is technically not a ‘‘server’’, since it has no cache and does not answer queries. A host which has an */etc/resolv.conf* file listing only remote hosts, and which does not run a name server of its own, is sometimes called a Remote Server but more often it is called simply a DNS Client.

4.3. Slave Server

A Slave Server is a server that always forwards queries it cannot satisfy from its cache, to a fixed list of *forwarding* servers instead of interacting with the master nameservers for the root and other domains. The queries to the *forwarding servers* are recursive queries. There may be one or more forwarding servers, and they are tried in turn until the list is exhausted. A Slave and forwarder configuration is typically used when you do not wish all the servers at a given site to be interacting with the rest of the Internet servers. A typical scenario would involve a number of workstations and a departmental timesharing machine with Internet access. The workstations might be administratively prohibited from having Internet access. To give the workstations the appearance of access to the Internet domain system, the workstations could be Slave servers to the timesharing machine which would forward the queries and interact with other nameservers to resolve the query before returning the answer. An added benefit of using the forwarding feature is that the central machine develops a much more complete cache of information that all the workstations can take advantage of. The use of Slave mode and forwarding is discussed further under the description of the named bootfile commands.

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

5. Setting up Your Own Domain

When setting up a domain that is going to be on a public network the site administrator should contact the organization in charge of the network and request the appropriate domain registration form. An organization that belongs to multiple networks (such as the *Internet* and *BITNET*) should register with only one network.

The contacts are as follows:

5.1. Internet

Sites on the Internet who need information on setting up a domain should contact the registrar for their network, which is one of the following:

MILnet	HOSTMASTER@NIC.DDN.MIL
other	HOSTMASTER@RS.INTERNIC.NET

You may also want to be placed on the BIND mailing list, which is a mail group for people on the Internet who run BIND. The group discusses future design decisions, operational problems, and other related topic. The address to request being placed on this mailing list is:

bind-request@uunet.uu.net

5.2. BITNET

If you are on the BITNET and need to set up a domain, contact INFO@BITNIC.

5.3. Subdomains of Existing Domains

If you want a subdomain of some existing domain, you should find the contact point for the parent domain rather than asking one of the above top-level registrars. There should be a convention that **registrar**@*domain* or **hostmaster**@*domain* for any given domain will always be an alias for that domain's registrar (somewhat analogous to **postmaster**), but there is no such convention. Try it as a last resort, but first you should examine the *SOA* record for the domain and send mail to the ‘‘responsible person’’ shown therein.

6. Files

The name server uses several files to load its data base. This section covers the files and their formats needed for *named*.

6.1. Boot File

This is the file that is first read when *named* starts up. This tells the server what type of server it is, which zones it has authority over and where to get its initial data. The default location for this file is */etc/named.boot*. However this can be changed by setting the *BOOTFILE* variable when you compile *named* or by specifying the location on the command line when *named* is started up.

6.1.1. Domain

A default domain may be specified for the nameserver using a line such as

domain Berkeley.Edu

Older name servers use this information when they receive a query for a name without a ‘‘**.**’’ that is not known. Newer designs assume that the resolver library will append its own idea of a ‘‘default domain’’ to any unqualified names. Though the name server can still be compiled with support for the *domain* directive in the boot file, the default is to leave it out and we strenuously recommend against its use. If you use this feature, clients outside your local domain which send you requests about unqualified names will have the implicit qualification of your domain rather than theirs. The proper place for this function is on the client, in their **/etc/resolv.conf** (or equivalent) file. Use of the *domain* directive in your boot file is strongly discouraged.

6.1.2. Directory

The *directory* directive specifies the directory in which the nameserver should run, allowing the other file names in the boot file to use relative path names. There can be only one *directory* directive and it should be given before any other directives that specify file names.

directory /var/named

If you have more than a couple of named files to be maintained, you may wish to place the named files in a directory such as /var/named and adjust the directory command properly. The main purposes of this command are to make sure named is in the proper directory when trying to include files by relative path names with $Include and to allow named to run in a location that is reasonable to dump core if it feels the urge.

6.1.3. Primary Service

The line in the boot file that designates the server as a primary server for a zone looks as follows:

primary Berkeley.Edu ucbhosts

The first field specifies that the server is a primary one for the zone stated in the second field. The third field is the name of the file from which the data is read.

The above assumes that the zone you are specifying is a class *IN* zone. If you wish to designate a different class you can append */class* to the first field, where *class* is either the integer value or the standard mnemonic for the class. For example the line for a primary server for a hesiod class zone looks as follows:

primary/HS Berkeley.Edu hesiod.data

Note that this support for specifying other than class *IN* zones is a compile-time option which your vendor may not have enabled when they built your operating system.

6.1.4. Secondary Service

The line for a secondary server is similar to the primary except that it lists addresses of other servers (usually primary servers) from which the zone data will be obtained.

secondary *Berkeley.Edu 128.32.0.10 128.32.0.4 ucbhosts.bak*

The first field specifies that the server is a secondary master server for the zone stated in the second field. The two network addresses specify the name servers which have data for the zone. Note that at least one of these will be a *primary*, and, unless you are using some protocol other than IP/DNS for your zone transfer mechanism, the others will all be other *secondary* servers. Having your secondary server pull data from other secondary servers is usually unwise, since you can add delay to the propagation of zone updates if your network's connectivity varies in pathological but common ways. The intended use for multiple addresses on a *secondary* declaration is when the *primary* server has multiple network interfaces and therefore multiple host addresses. The secondary server gets its data across the network from one of the listed servers. The server addresses are tried in the order listed. If a filename is present after the list of primary servers, data for the zone will be dumped into that file as a backup. When the server is first started, the data is loaded from the backup file if possible, and a primary server is then consulted to check that the zone is still up-to-date. Note that listing your server as a *secondary* server does not neccessarily make it one — the parent zone must *delegate* authority to your server as well as the primary and the other secondaries, or you will be transferring a zone over for no reason; no other server will have a reason to query you for that zone unless the parent zone lists you as a server for the zone.

As with primary you may specify a secondary server for a class other than *IN* by appending */class* to the *secondary* keyword, e.g., *secondary/HS*.

6.1.5. Stub Service

The line for a stub server is similar to a secondary.

stub *Berkeley.Edu* *128.32.0.10 128.32.0.4 ucbhosts.bak*

The first field specifies that the server is a stub server for the zone stated in the second field.

Stub zones are intened to ensure that a primary for a zone always has the correct nameserver records for children of that zone. If the primary is not a secondary for a child zone it should be configured with stub zones for all its children. Stub zones provide a mechanism to allow nameserver records for a zone to be specified in only one place.

primary	*CSIRO.AU*	*csiro.dat*	
stub	*dms.CSIRO.AU*	*130.155.16.1*	*dms.stub*
stub	*dap.CSIRO.AU*	*130.155.98.1*	*dap.stub*

6.1.6. Caching Server

You do not need a special line to designate that a server is a caching server. What denotes a ‘‘caching only’’ server is the absence of authority lines, such as *secondary* or *primary* in the boot file.

All servers, including ‘‘caching only’’ servers, should have a line as follows in the boot file to prime the name servers cache:

cache . *root.cache*

All cache files listed will be read in at named boot time and any values still valid will be reinstated in the cache and the root nameserver information in the cache files will be used until a root query is actually answered by one of the name servers in your cache file, at which time that answer will be used until it times out and your cache file will be ignored.

As with *primary* and *secondary*, you may specify a secondary server for a class other than *IN* by appending */class* to the *cache* keyword, e.g., *class/HS*.

Do not put anything into your *cache* files other than root server information.

6.1.7. Forwarders

Any server can make use of *forwarders*. A *forwarder* is another server capable of processing recursive queries that is willing to try resolving queries on behalf of other systems. The *forwarders* command specifies forwarders by internet address as follows:

forwarders *128.32.0.10 128.32.0.4*

There are two main reasons for wanting to do so. First, some systems may not have full network access and may be prevented from sending any IP packets into the rest of the Internet and therefore must rely on a forwarder which does have access to the full net. The second reason is that the forwarder sees a union of all queries as they pass through his server and therefore it builds up a very rich cache of data compared to the cache in a typical workstation nameserver. In effect, the *forwarder* becomes a meta-cache that all hosts can benefit from, thereby reducing the total number of queries from that site to the rest of the net.

The effect of ''forwarders'' is to prepend some fixed addresses to the list of name servers to be tried for every query. Normally that list is made up only of higher-authority servers discovered via *NS* record lookups for the relevant domain. If the forwarders do not answer, then unless the *slave* directive was given, the appropriate servers for the domains will be queried directly.

6.1.8. Slave Servers

Slave mode is used if the use of forwarders is the only possible way to resolve queries due to lack of full net access or if you wish to prevent the nameserver from using other than the listed forwarders. Slave mode is activated by placing the simple command

slave

in the bootfile. If *slave* is used, then you must specify forwarders. When in slave mode, the server will forward each query to each of the the forwarders until an answer is found or the list of forwarders is exhausted. The server will not try to contact any remote name server other than those named in the *forwarders* list.

So while *forwarders* adds to the end of the ''server list'' for each query, *slave* causes the ''server list'' to contain *only* those addresses listed in the *forwarders* declarations. Careless use of the *slave* directive can cause really horrible forwarding loops, since you could end up forwarding queries only to some set of hosts which are also slaves, and one or several of them could be forwarding queries back to you.

Use of the *slave* directive should be considered very carefully.

6.1.9. Zone Transfer Restrictions

It may be the case that your organization does not wish to give complete lists of your hosts to anyone on the Internet who can reach your name servers. While it is still possible for people to ''iterate'' through your address range, looking for *PTR* records, and build a list of your hosts the ''slow'' way, it is still considered reasonable to restrict your export of zones via the zone transfer protocol. To limit the list of neighbors who can transfer zones from your server, use the

xfrnets

directive. This directive has the same syntax as *forwarders* except that you can list network numbers in addition to host addresses. For example, you could add the directive *xfrnets 16.0.0.0* if you wanted to permit only hosts on Class A network number 16 to transfer zones from your server. This is not nearly granular enough, and a future version of BIND will permit such access-control to be specified on a per-zone basis rather than the current ''global'' basis.

The *xfrnets* directive may also be given as *tcplist* for compatibility with interim releases of BIND 4.9.

Note that *xfrnets* support is a compile-time option which your vendor may not have enabled when they built your operating system.

6.1.10. Sorting Addresses

If there are multiple addresses available for a name server which BIND wants to contact, BIND will try the ones it believes are ‘‘closest’’ first. ‘‘Closeness’’ is defined in terms of similarity-of-address; that is, if one address is on the same *subnet* as some interface of the local host, then that address will be tried first. Failing that, an address which is on the same *network* will be tried first. Failing that, they will be tried in a more-or-less random order unless the *sortlist* directive was given in the *named.boot* file. *sortlist* has a syntax similar to *forwarders* and *xfrnets*; you give it a list of networks and it uses these to ‘‘prefer’’ some remote name server addresses over others. If you are on a Class C net which has a Class B net between you and the rest of the Internet, you could try to improve the name server's luck in getting answers by listing the Class B network's number in a *sortlist* directive. This should have the effect of trying ‘‘closer’’ servers before the more ‘‘distant’’ ones. Note that this behaviour is new in BIND 4.9.<

The other and older effect of the *sortlist* directive is to cause BIND to sort the *A* records in any response it generates, so as to put those which appear on the *sortlist* earlier than those which do not. This is not as helpful as you might think, since many clients will reorder the *A* records either at random or using LIFO.

In actual practice, noone uses this directive since it hardwires information which changes rapidly; a network which is ‘‘close’’ today may be ‘‘distant’’ next month. Since BIND builds up a cache of the remote name servers' response times, it will quickly converge on ‘‘reasonable’’ behaviour, which isn't the same as ‘‘optimal’’ but it's close enough. Future directions for BIND include choosing addresses based on local interface metrics (on hosts which have more than one) and perhaps on routing table information. We do not intend to solve the generalized ‘‘multi-homed host’’ problem, but we should be able to do a little better than we're doing now. Likewise, we hope to see a higher-level resolver library that sorts responses using topology information that only exists on the client's host.

6.1.11. Bogus Name Servers

It happens occasionally that some remote name server goes ‘‘bad’’. You can tell your name server to refuse to listen to or ask questions of certain other name servers by listing them in a *bogusns* directive in your *named.boot* file. Its syntax is the same as *forwarders* — you just give it a list of dotted-quad Internet addresses.

Note that *bogusns* support is a compile-time option which your vendor may not have enabled when they built your operating system.

6.1.12. Segmented Boot Files

If you are secondary for a lot of zones, you may find it convenient to split your *named.boot* file into a static portion which hardly ever changes (directives such as *directory*, *sortlist*, *xfrnets* and *cache* could go here), and dynamic portions that change frequently (all of your *primary* directives might go in one file, and all of your *secondary* directives might go in another file — and either or both of these might be fetched automatically from some neighbor so that they can change your list of secondary zones without requiring your active intervention). You can accomplish this via the *include* directive, which takes just a single file name as its argument. No quotes are needed around the file name. The file name will be evaluated after the name server has changed its working directory to that specified in the *directory* directive, so you can use relative pathnames if your system supports them.

6.2. Resolver Configuration

The resolver will try to contact a nameserver on the localhost if it cannot find its configuration file. You should install the configuration file on every host anyway, since you can list the local host's address if the localhost runs a nameserver, and there is no other recommended way to specify a system-level default domain. Note that if you wish to list the local host in your resolver configuration file, you should probably use its primary Internet address rather than a localhost alias such as 127.0.0.1 or 0.0.0.0. This is due to a bug in the handling of connected SOCK_DGRAM sockets in some versions of the BSD networking code. If you must use an address-alias, you should prefer 0.0.0.0 (or simply ‘‘0’’) over 127.0.0.1, though be warned that depending on the vintage of your BSD-derived networking code, both of them are capable of failing in their own ways.

The configuration file’s name is */etc/resolv.conf*. This file designates the name servers on the network that should be sent queries. It is considered reasonable to create this file even if you run a local server, since its contents will be cached by each client of the resolver library when the client makes its first call to a resolver routine. If you run a name server locally, list it in your *resolv.conf* file.

The *resolv.conf* file contains directives, one per line, of the following forms:

> ; comment
> # another comment
> domain *local-domain*
> search *search-list*
> nameserver *server-address*
> sortlist *sort-list*
> options *option-list*

Exactly one of the *domain* or *search* directives should be given, exactly once. If the *search* directive is given, the first item in the given *search-list* will override any previously-specified *local-domain*. The *nameserver* directive may be given up to three times; additional *nameserver* directives will be ignored. Comments may be given by starting a line with a ‘‘;’’ or ‘‘#’’; note that comments were not permitted in versions of the resolver earlier than the one included with BIND 4.9 — so if your vendor’s resolver supports comments, you know they are really on the ball.

The *local-domain* will be appended to any query-name that does not contain a ‘‘.’’. *local-domain* can be overridden on a per-process basis by setting the LOCALDOMAIN environment variable. Note that *local-domain* processing can be disabled by setting an option in the resolver.

The *search-list* is a list of domains which are tried, in order, as qualifying domains for query-names which do not contain a ‘‘.’’. Note that *search-list* processing can be disabled by setting an option in the resolver. Also note that the environment variable ‘‘LOCALDOMAIN’’ can override this *search-list* on a per-process basis.

The *server-address*’s are aggregated and then used as the default destination of queries generated through the resolver. This is, in other words, the way you tell the resolver which name servers it should use. It is possible for a given client application to override this list, and this is often done inside the name server (which is itself a *resolver* client) and in test programs such as *nslookup*.

The *sort-list* is a list of IP address, netmask pairs. Addresses returned by gethostbyname are sorted to the order specifed by this list. Any addresses that do not match the address netmask pair will returned after those that do. The netmask is optional and the natural netmask will be used if not specified.

The *option-list* is a list of options which each override some internal resolver variable. Supported options at this time are:

debug
: sets the RES_DEBUG bit in **_res.options**.

ndots:*n*
: sets the lower threshold (measured in ‘‘number of dots’’) on names given to *res_query*() such that names with more than this number of dots will be tried as absolute names before any *local-domain* or *search-list* processing is done. The default for this internal variable is ‘‘1’’.

Finally, if the environment variable HOSTALIASES is set, it is taken to contain the name of a file which in turn contains resolver-level aliases. These aliases are applied only to names which do not contain any ‘‘.’’ characters, and they are applied to query-names before the query is generated. Note that the resolver options governing the operation of *local-domain* and *search-list* do not apply to HOSTALIASES.

6.3. Cache Initialization

6.3.1. root.cache

The name server needs to know the servers that are the authoritative name servers for the root domain of the network. To do this we have to prime the name server's cache with the addresses of these higher authorities. The location of this file is specified in the boot file. This file uses the Standard Resource Record Format (aka. Masterfile Format) covered further on in this paper.

6.3.2. named.local

This file specifies the *PTR* record for the local loopback interface, better known as *localhost*, whose network address is 127.0.0.1. The location of this file is specified in the boot file. It is vitally important to the proper operation of every name server that the 127.0.0.1 address have a *PTR* record pointing back to the name ''**localhost.***my.dom.ain*''. The name of this *PTR* record is always ''**1.0.0.127.IN-ADDR.ARPA**''. This is neccessary if you want your users to be able to use hostname-authentication (*hosts.equiv* or *~/.rhosts*) on the name ''**localhost**''. As implied by this *PTR* record, there should be an *A* record in your domain specifying that ''**localhost.***my.dom.ain*'' has the Internet address 127.0.0.1.

6.4. Domain Data Files

There are two standard files for specifying the data for a domain. These are *hosts* and *host.rev*. These files use the Standard Resource Record Format covered later in this paper. Note that the file names are arbitrary; many network administrators prefer to name their zone files after the domains they contain, especially in the average case which is where a given server is primary and/or secondary for many different zones.

6.4.1. hosts

This file contains all the data about the machines in this zone. The location of this file is specified in the boot file.

6.4.2. hosts.rev

This file specifies the IN-ADDR.ARPA domain. This is a special domain for allowing address to name mapping. As internet host addresses do not fall within domain boundaries, this special domain was formed to allow inverse mapping. The IN-ADDR.ARPA domain has four labels preceding it. These labels correspond to the 4 octets of an Internet address. All four octets must be specified even if an octets is zero. The Internet address 128.32.0.4 is located in the domain 4.0.32.128.IN-ADDR.ARPA. This reversal of the address is awkward to read but allows for the natural grouping of hosts in a network.

6.5. Standard Resource Record Format

The records in the name server data files are called resource records. The Standard Resource Record Format (RR) is specified in RFC1035. The following is a general description of these records:

{name} *{ttl}* *addr-class* *Record Type* *Record Specific data*

Resource records have a standard format shown above. The first field is always the name of the domain record and it must always start in column 1. For all RR's other than the first in a file, the name may be left blank; in that case it takes on the name of the previous RR. The second field is an optional time to live field. This specifies how long this data will be stored in the data base. By leaving this field blank the default time to live is specified in the *Start Of Authority* resource record (see below). The third field is the address class; currently, only one class is supported: *IN* for internet addresses and other internet information. Limited support is included for the *HS* class, which is for MIT/Athena ‘‘Hesiod’’ information. The fourth field states the type of the resource record. The fields after that are dependent on the type of the RR. Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server data base are case insensitive.

The following characters have special meanings:

‘‘**.**’’ A free standing dot in the name field refers to the current domain.

‘‘@’’ A free standing @ in the name field denotes the current origin.

‘‘**..**’’ Two free standing dots represent the null domain name of the root when used in the name field.

‘‘\X’’ Where X is any character other than a digit (0-9), quotes that character so that its special meaning does not apply. For example, ‘‘\.’’ can be used to place a dot character in a label.

‘‘\DDD’’
Where each D is a digit, is the octet corresponding to the decimal number described by DDD. The resulting octet is assumed to be text and is not checked for special meaning.

‘‘( )’’ Parentheses are used to group data that crosses a line. In effect, line terminations are not recognized within parentheses.

‘‘;’’ Semicolon starts a comment; the remainder of the line is ignored.

‘‘*’’ An asterisk signifies wildcarding. Note that this is just another data character whose special meaning comes about only during internal name server search operations. Wildcarding is only meaningful for some RR types (notably *MX*), and then only in the name field — not in the data fields.

Anywhere a name appears — either in the name field or in some data field defined to contain names — the current origin will be appended if the name does not end in a ‘‘**.**’’. This is useful for appending the current domain name to the data, such as machine names, but may cause problems where you do not want this to happen. A good rule of thumb is that, if the name is not in the domain for which you are creating the data file, end the name with a ‘‘**.**’’.

6.5.1. $INCLUDE

An include line begins with $INCLUDE, starting in column 1, and is followed by a file name, and, optionally, by a new temporary $ORIGIN to be used while reading this file. This feature is particularly useful for separating different types of data into multiple files. An example would be:

$INCLUDE /usr/local/adm/named/data/mail-exchangers

The line would be interpreted as a request to load the file */usr/named/data/mail-exchangers*. The $INCLUDE command does not cause data to be loaded into a different zone or tree. This is simply a way to allow data for a given primary zone to be organized in separate files. Not even the ‘‘temporary $ORIGIN’’ feature described above is sufficient to cause your data to branch out into some other zone — zone boundaries can only be introduced in the boot file.

6.5.2. ‘‘$ORIGIN’’

The origin is a way of changing the origin in a data file. The line starts in column 1, and is followed by a domain origin. This seems like it could be useful for putting more then one zone into a data file, but that’s not how it works. The name server fundamentally requires that a given zone map entirely to some specific file. You should therefore be very careful to use $ORIGIN only once at the top of a file, or, within a file, to change to a ‘‘lower’’ domain in the zone — never to some other zone altogether.

6.5.3. SOA - Start Of Authority

name	*{ttl}*	*addr-class*	*SOA*	*Origin*	*Person in charge*
@		IN	SOA	ucbvax.Berkeley.Edu.	kjd.ucbvax.Berkeley.Edu. (
			1993041403	; Serial	
			10800	; Refresh	
			1800	; Retry	
			3600000	; Expire	
			259200)	; Minimum	

The *Start of Authority, SOA*, record designates the start of a zone. The name is the name of the zone. Origin is the name of the host on which this data file resides. Person in charge is the mailing address for the person responsible for the name server. The serial number is the version number of this data file; this number should be incremented whenever a change is made to the data. Older servers permitted the use of a phantom ‘‘.’’ in this and other numbers in a zone file; the meaning of n.m was ‘‘n000m’’ rather than the more intuitive ‘‘n*1000+m’’ (such that 1.234 translated to 1000234 rather than to 1234). This feature has been deprecated due to its obscurity, unpredictability, and lack of neccessity. Note that using a ‘‘YYYYMMDDNN’’ notation you can still make 100 changes per day until the year 4294. You should choose a notation that works for you. If you’re a clever *perl* programmer you could even use *RCS* version numbers to help generate your zone serial numbers. The refresh indicates how often, in seconds, the secondary name servers are to check with the primary name server to see if an update is needed. The retry indicates how long, in seconds, a secondary server should wait before retrying a failed zone transfer. Expire is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. Minimum is the default number of seconds to be used for the Time To Live field on resource records which do not specify one in the zone file. It is also an enforced minimum on Time To Live if it is specified on an RR. There should only be one *SOA* record per zone.

6.5.4. NS - Name Server

{name}	*{ttl}*	*addr-class*	*NS*	*Name servers name*
		IN	NS	ucbarpa.Berkeley.Edu.

The *Name Server* record, *NS*, lists a name server responsible for a given domain. The first name field lists the domain that is serviced by the listed name server. There should be one *NS* record for each name server for the domain, and every domain should have at least two nameservers.

6.5.5. A - Address

{name}	*{ttl}*	*addr-class*	*A*	*address*
ucbarpa		IN	A	128.32.0.4
		IN	A	10.0.0.78

The *Address* record, *A*, lists the address for a given machine. The name field is the machine name and the address is the network address. There should be one *A* record for each address of the machine.

6.5.6. HINFO - Host Information

{name}	*{ttl}*	*addr-class*	*HINFO*	*Hardware*	*OS*
		IN	HINFO	VAX-11/780	UNIX

Host Information resource record, *HINFO*, is for host specific data. This lists the hardware and operating system that are running at the listed host. If you want to include a space in the machine name you must quote the name. There could be one *HINFO* record for each host, though for security reasons most domains don’t have any *HINFO* records at all. No application depends on them.

6.5.7. WKS - Well Known Services

{name}	*{ttl}*	*addr-class*	*WKS*	*address*	*protocol*	*list of services*
		IN	WKS	128.32.0.10	UDP	who route timed domain
		IN	WKS	128.32.0.10	TCP	(echo telnet discard sunrpc sftp uucp-path systat daytime netstat qotd nntp link chargen ftp auth time whois mtp pop rje finger smtp supdup hostnames domain nameserver)

The *Well Known Services* record, *WKS*, describes the well known services supported by a particular protocol at a specified address. The list of services and port numbers come from the list of services specified in */etc/services.* There should be only one *WKS* record per protocol per address. Note that RFC 1123 says of *WKS* records:

```
2.2  Using Domain Name Service
...
  An application SHOULD NOT rely on the ability to locate a WKS
  record containing an accurate listing of all services at a
  particular host address, since the WKS RR type is not often used
  by Internet sites.  To confirm that a service is present, simply
  attempt to use it.
...
  5.2.12  WKS Use in MX Processing: RFC-974, p. 5

    RFC-974 [SMTP:3] recommended that the domain system be queried
    for WKS ("Well-Known Service") records, to verify that each
    proposed mail target does support SMTP.  Later experience has
    shown that WKS is not widely supported, so the WKS step in MX
    processing SHOULD NOT be used.
...
    6.1.3.6  Status of RR Types
...
          The TXT and WKS RR types have not been widely used by
          Internet sites; as a result, an application cannot rely
          on the the existence of a TXT or WKS RR in most
          domains.
```

6.5.8. CNAME - Canonical Name

aliases	*{ttl}*	*addr-class*	*CNAME*	*Canonical name*
ucbmonet		IN	CNAME	monet

The *Canonical Name* resource record, *CNAME*, specifies an alias or nickname for the official, or canonical, host name. This record should be the only one associated with the alias name. All other resource records should be associated with the canonical name, not with the nickname. Any resource records that include a domain name as their value (e.g., NS or MX) *must* list the canonical name, not the nickname.

Nicknames are also useful when a host changes its name. In that case, it is usually a good idea to have a *CNAME* record so that people still using the old name will get to the right place.

6.5.9. PTR - Domain Name Pointer

name	*{ttl}*	*addr-class*	*PTR*	*real name*
7.0		IN	PTR	monet.Berkeley.Edu.

A *Domain Name Pointer* record, *PTR,* allows special names to point to some other location in the domain. The above example of a *PTR* record is used in setting up reverse pointers for the special *IN-ADDR.ARPA* domain. This line is from the example *hosts.rev* file. *PTR* records are needed by the *gethostbyaddr* function. Note the trailing ''.'' which prevents BIND from appending the current $ORIGIN.

6.5.10. MX - Mail Exchanger

name	*{ttl}*	*addr-class*	*MX*	*preference value*	*mail exchanger*
Munnari.OZ.AU.		IN	MX	0	Seismo.CSS.GOV.
*.IL.		IN	MX	0	RELAY.CS.NET.

Mail eXchanger records, *MX*, are used to specify a list of hosts which are configured to receive mail sent to this domain name. Every name which receives mail should have an *MX* since if one is not found at the time mail is being delivered, an *MX* will be ''imputed'' with a cost of 0 and a destination of the host itself. If you want a host to receive its own mail, you should create an *MX* for your host's name, pointing at your host's name. It is better to have this be explicit than to let it be imputed by remote mailers. In the first example, above, Seismo.CSS.GOV. is a mail gateway that knows how to deliver mail to Munnari.OZ.AU.. These two machines may have a private connection or use a different transport medium. The preference value is the order that a mailer should follow when there is more then one way to deliver mail to a single machine. Note that lower numbers indicate higher precedence, and that mailers are supposed to randomize same-valued *MX* hosts so as to distribute the load evenly if the costs are equal. See RFC 974 for more detailed information.

Wildcard names containing the character ''*'' may be used for mail routing with *MX* records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. Second example, above, all mail to hosts in the domain IL is routed through RELAY.CS.NET. This is done by creating a wildcard resource record, which states that *.IL has an *MX* of RELAY.CS.NET. Wildcard *MX* records are not very useful in practice, though, since once a mail message gets to the gateway for a given domain it still has to be routed *within* that domain and it is not currently possible to have an apparently-different set of *MX* records inside and outside of a domain. If you won't be needing any Mail Exchangers inside your domain, go ahead and use a wildcard. If you want to use both wildcard ''top-level'' and specific ''interior'' *MX* records, note that each specific record will have to ''end with'' a complete recitation of the same data that is carried in the top-level record. This is because the specific *MX* records will take precedence over the top-level wildcard records, and must be able to perform the top-level's if a given interior domain is to be able to receive mail from outside the gateway. Wildcard *MX* records are very subtle and you should be careful with them.

6.5.11. TXT - Text

name	*{ttl}*	*addr-class*	*TXT*	*string*
Munnari.OZ.AU.		IN	TXT	"foo"

A *TXT* record contains free-form textual data. The syntax of the text depends on the domain where it is found; several systems use *TXT* records to encode the local user database (*/etc/passwd*) and other administrative data. MIT Hesiod is one such system, which, though it uses an addr-class of *HS* rather than *IN*, implements its database with *TXT* records in the DNS.

6.5.12. RP - Responsible Person

owner	*{ttl}*	*addr-class*	*RP*	*mbox-domain-name*	*TXT-domain-name*
franklin		IN	RP	ben.franklin.berkeley.edu.	sysadmins.berkeley.edu.

The Responsible Person record, *RP*, identifies the name or group name of the responsible person for a host. Often it is desirable to be able to identify the responsible entity for a particular host. When that host is down or malfunctioning, you would want to contact those parties who might be able to repair the host.

The first field, *mbox-domain-name*, is a domain name that specifies the mailbox for the responsible person. Its format in master files uses the DNS convention for mailbox encoding, identical to that used for the *Person-in-charge* mailbox field in the SOA record. In the example above, the mbox domain name shows the encoding for ‘‘**<ben@franklin.berkeley.edu>**’’. The root domain name (just ‘‘**.**’’) may be specified to indicate that no mailbox is available.

The second field, *TXT-domain-name*, is a domain name for which *TXT* records exist. A subsequent query can be performed to retrieve the associated *TXT* resource records at *TXT* domain name. This provides a level of indirection so that the entity can be referred to from multiple places in the DNS. The root domain name (just ‘‘**.**’’) may be specified for TXT domain name to indicate that no associated *TXT* RR exists. In the example above, ‘‘**sysadmins.berkeley.edu.**’’ is the name of a TXT record that might contain some text with names and phone numbers.

The format of the *RP* record is class-insensitive. Multiple *RP* records at a single name may be present in the database, though they should have identical TTLs.

The *RP* record is still experimental; not all name servers implement or recognize it.

6.5.13. AFSDB - DCE or AFS Server

name	*{ttl}*	*addr-class*	*AFSDB*	*subtype*	*mail exchanger*
toaster.com.		IN	AFSDB	1	jack.toaster.com
toaster.com.		IN	AFSDB	1	jill.toaster.com.
toaster.com.		IN	AFSDB	2	tracker.toaster.com.

AFSDB records are used to specify the hosts that provide a style of distributed service advertised under this domain name. A subtype value (analogous to the ‘‘preference’’ value in the *MX* record) indicates which style of distributed service is provided with the given name. Subtype 1 indicates that the named host is an AFS (R) database server for the AFS cell of the given domain name. Subtype 2 indicates that the named host provides intra-cell name service for the DCE (R) cell named by the given domain name. In the example above, jack**.**toaster**.**com and jill**.**toaster**.**com are declared to be AFS database servers for the toaster**.**com AFS cell, so that AFS clients wishing service from tracker**.**com are directed to those two hosts for further information. The third record declares that tracker**.**toaster**.**com houses a directory server for the root of the DCE cell toaster**.**com, so that DCE clients that wish to refer to DCE services should consult with the host tracker**.**toaster**.**com for further information. The DCE sub-type of record is usually accompanied by a *TXTP record for other information specifying other details to be used in accessing the DCE cell. RFC 1183 contains more detailed information on the use of this record type.*

The *AFSDB* record is still experimental; not all name servers implement or recognize it.

6.6. Discussion about the TTL

The Time To Live assigned to the records and to the zone via the Minimum field in the SOA record is very important. High values will lead to lower BIND network traffic and faster response time. Lower values will tend to generate lots of requests but will allow faster propagation of changes.

Only changes and deletions from the zone are affected by the TTLs. Additions propagate according to the Refresh value in the SOA.

Experience has shown that sites use default TTLs for their zones varying from around 0.5 day to around 7 days. You may wish to consider boosting the default TTL shown in former versions of this guide from one day (86400 seconds) to three days (259200 seconds). This will drastically reduce the number of requests made to your name servers.

If you need fast propagation of changes and deletions, it might be wise to reduce the Minimum field a few days before the change, then do the modification itself and augment the TTL to its former value.

If you know that your zone is pretty stable (you mainly add new records without changing regularly old ones) then you may even wish to consider a TTL higher than three days.

Note that in any case, it makes no sense to have records with a TTL below the SOA Refresh delay, as Delay is the time required for secondaries to get a copy of the newly modified zone.

6.7. Sample Files

The following section contains sample files for the name server. This covers example boot files for the different types of servers and example domain data base files.

6.7.1. Boot Files

6.7.1.1. Primary Server

```
;
; Boot file for Primary Name Server
;

; type       domain                 source file or host
;
directory    /usr/local/adm/named
primary      Berkeley.Edu           ucbhosts
primary      32.128.in-addr.arpa    ucbhosts.rev
primary      0.0.127.in-addr.arpa   named.local
cache        .                      root.cache
```

6.7.1.2. Secondary Server

```
;
; Boot file for Secondary Name Server
;

; type       domain                 source file or host
;
directory    /usr/local/adm/named
secondary    Berkeley.Edu           128.32.0.4 128.32.0.10 ucbhosts.bak
secondary    32.128.in-addr.arpa    128.32.0.4 128.32.0.10 ucbhosts.rev.bak
primary      0.0.127.in-addr.arpa   named.local
cache        .                      root.cache
```

6.7.1.3. Caching Only Server

```
;
; Boot file for Caching Only Name Server
;

; type        domain                 source file or host
;
directory     /usr/local/adm/named
cache         .                      root.cache
primary       0.0.127.in-addr.arpa   named.local
```

6.7.2. Remote Server / DNS Client

6.7.2.1. /etc/resolv.conf

```
domain Berkeley.Edu
nameserver 128.32.0.4
nameserver 128.32.0.10
sortlist 130.155.160.0/255.255.240.0 130.155.0.0
```

6.7.3. root.cache

```
;
; Initial cache data for root domain servers.
;
; This data was current as of April 15, 1993.  The official and current
; version is always available from via anonymous FTP from RS.INTERNIC.NET
; as /domain/named.cache.
;
; Thanks to Long-Morrow@CS.Yale.EDU for providing this update.
;

.                       99999999   IN   NS   NS.NIC.DDN.MIL.
                        99999999   IN   NS   NS.NASA.GOV.
                        99999999   IN   NS   KAVA.NISC.SRI.COM.
                        99999999   IN   NS   TERP.UMD.EDU.
                        99999999   IN   NS   AOS.ARL.ARMY.MIL.
                        99999999   IN   NS   C.NYSER.NET.
                        99999999   IN   NS   NIC.NORDU.NET.
                        99999999   IN   NS   NS.INTERNIC.NET.

;  Prep the cache (hotwire the addresses).
NS.NIC.DDN.MIL.         99999999   IN   A    192.112.36.4
NS.NASA.GOV.            99999999   IN   A    128.102.16.10
NS.NASA.GOV.            99999999   IN   A    192.52.195.10
KAVA.NISC.SRI.COM.      99999999   IN   A    192.33.33.24
AOS.ARL.ARMY.MIL.       99999999   IN   A    128.63.4.82
AOS.ARL.ARMY.MIL.       99999999   IN   A    192.5.25.82
C.NYSER.NET.            99999999   IN   A    192.33.4.12
TERP.UMD.EDU.           99999999   IN   A    128.8.10.90
NIC.NORDU.NET.          99999999   IN   A    192.36.148.17
NS.INTERNIC.NET.        99999999   IN   A    198.41.0.4
```

6.7.4. named.local

```
@    IN   SOA   ucbvax.Berkeley.Edu.      kjd.ucbvax.Berkeley.Edu. (
                1993050201                ; Serial
                10800                     ; Refresh
                1800                      ; Retry
                3600000                   ; Expire
                259200 )                  ; Minimum
     IN   NS    ucbvax.Berkeley.Edu.      ; pedantic
1    IN   PTR   localhost.Berkeley.Edu.
```

6.7.5. Hosts

```
;
;   @(#)ucb-hosts   1.2   (berkeley)   88/02/05
;
@           IN   SOA     ucbvax.Berkeley.Edu.     kjd.monet.Berkeley.Edu. (
                         1988020501               ; Serial
                         10800                    ; Refresh
                         1800                     ; Retry
                         3600000                  ; Expire
                         259200 )                 ; Minimum
            IN   NS      ucbarpa.Berkeley.Edu.
            IN   NS      ucbvax.Berkeley.Edu.
localhost   IN   A       127.1
                         ; note that 127.1 is the same as 127.0.0.1; see inet(3n)
ucbarpa     IN   A       128.32.4
            IN   A       10.0.0.78
            IN   HINFO   VAX-11/780 UNIX
arpa        IN   CNAME   ucbarpa
ernie       IN   A       128.32.6
            IN   HINFO   VAX-11/780 UNIX
ucbernie    IN   CNAME   ernie
monet       IN   A       128.32.7
            IN   A       128.32.130.6
            IN   HINFO   VAX-11/750 UNIX
ucbmonet    IN   CNAME   monet
ucbvax      IN   A       10.2.0.78
                         ; 128.32.10 means 128.32.0.10; see inet(3n)
            IN   A       128.32.10
                         ; HINFO and WKS are widely unused,
                         ; but we'll show them as examples.
            IN   HINFO   VAX-11/750 UNIX
            IN   WKS     128.32.0.10 TCP ( echo telnet
                         discard sunrpc sftp
                         uucp-path systat daytime
                         netstat qotd nntp
                         link chargen ftp
                         auth time whhois mtp
                         pop rje finger smtp
                         supdup hostnames
                         domain
                         nameserver )
vax         IN   CNAME   ucbvax
toybox      IN   A       128.32.131.119
            IN   HINFO   Pro350 RT11
toybox      IN   MX      0  monet.Berkeley.Edu.
csrg        IN   MX      0 Ralph.CS
            IN   MX      0 Zhou.CS
            IN   MX      0 Painter.CS
            IN   MX      0 Riggle.CS
            IN   MX      0 Terry.CS
            IN   MX      0 Kevin.CS
```

6.7.6. host.rev

```
;
;	@(#)ucb-hosts.rev	1.1	(Berkeley)	86/02/05
;
@	IN	SOA	ucbvax.Berkeley.Edu.	kjd.monet.Berkeley.Edu. (
			1986020501	; Serial
			10800	; Refresh
			1800	; Retry
			3600000	; Expire
			259200 )	; Minimum
	IN	NS	ucbarpa.Berkeley.Edu.
	IN	NS	ucbvax.Berkeley.Edu.
0.0	IN	PTR	Berkeley-net.Berkeley.EDU.
	IN	A	255.255.255.0
0.130	IN	PTR	csdiv-net.Berkeley.EDU.
4.0	IN	PTR	ucbarpa.Berkeley.Edu.
6.0	IN	PTR	ernie.Berkeley.Edu.
7.0	IN	PTR	monet.Berkeley.Edu.
10.0	IN	PTR	ucbvax.Berkeley.Edu.
6.130	IN	PTR	monet.Berkeley.Edu.
```

7. Domain Management

This section contains information for starting, controlling and debugging *named*.

7.1. /etc/rc.local

The hostname should be set to the full domain style name in */etc/rc.local* using *hostname (1)*. The following entry should be added to */etc/rc.local* to start up *named* at system boot time:

```
if [ -f /etc/named ]; then
        /etc/named [options] & echo -n ' named'     >/dev/console
fi
```

This usually directly follows the lines that start *syslogd*. **Do Not** attempt to run *named* from *inetd*. This will continuously restart the name server and defeat the purpose of the cache.

7.2. /etc/named.pid

When *named* is successfully started up it writes its process id into the file */etc/named.pid*. This is useful to programs that want to send signals to *named*. The name of this file may be changed by defining *PIDFILE* to the new name when compiling *named*.

7.3. /etc/hosts

The *gethostbyname ()* library call can detect if *named* is running. If it is determined that *named* is not running it will look in */etc/hosts* to resolve an address. This option was added to allow *ifconfig (8C)* to configure the machines local interfaces and to enable a system manager to access the network while the system is in single user mode. It is advisable to put the local machines interface addresses and a couple of machine names and address in */etc/hosts* so the system manager can rcp files from another machine when the system is in single user mode. The format of */etc/hosts* has not changed. See *hosts (5)* for more information. Since the process of reading */etc/hosts* is slow, it is not advisable to use this option when the system is in multi user mode.

7.4. Signals

There are several signals that can be sent to the *named* process to have it do tasks without restarting the process.

7.4.1. Reload

SIGHUP - Causes *named* to read *named.boot* and reload the database. This is useful when you have made a change to a ‘‘primary’’ data file and you want *named*'s internal database to reflect the change. If you build BIND with the FORCED_RELOAD option, then SIGHUP also has the effect of scheduling all ‘‘secondary’’ zones for serial-number checks, which could lead to zone transfers ahead of the usual schedule. Normally serial-number compares are done only at the intervals specified in the zone’s SOA record.

7.4.2. Debugging

When *named* is running incorrectly, look first in */var/log/messages* and check for any messages logged by *syslog*. Next send it a signal to see what is happening. Unless you run it with the ‘‘-d’’ option, *named* has very little to say on its standard output or standard error. Everything *named* has to say, it says to *syslog*.

SIGINT - Dumps the current data base and cache to */var/ tmp/ named_dump . db* This should give you an indication to whether the data base was loaded correctly. The name of the dump file may be changed by defining *DUMPFILE* to the new name when compiling *named*.

Note: the following two signals only work when *named* is built with *DEBUG* defined.

SIGUSR1 - Turns on debugging. Each following USR1 increments the debug level. The output goes to */var/tmp/named.run* The name of this debug file may be changed by defining *DEBUGFILE* to the new name before compiling *named*.

SIGUSR2 - Turns off debugging completely.

For more detailed debugging, define DEBUG when compiling the resolver routines into */lib/libc.a*.

SIGWINCH - Toggles tracing of all incoming queries if *named* has been compiled with *QRYLOG* defined. The trace is sent to syslog, and is huge, but it is very useful for tracking down problems.

To run with tracing of all queries specify the *-q* flag on the command line. If you routinely log queries you will probably want to analyze the results using the dnsstats stats script in the contrib directory.

ACKNOWLEDGEMENTS

Many thanks to the users at U.C. Berkeley for falling into many of the holes involved with integrating BIND into the system so that others would be spared the trauma. I would also like to extend gratitude to Jim McGinness and Digital Equipment Corporation for permitting me to spend most of my time on this project.

Ralph Campbell, Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss and everyone else on the DARPA Internet who has contributed to the development of BIND. To the members of the original BIND project, Douglas Terry, Mark Painter, David Riggle and Songnian Zhou.

Anne Hughes, Jim Bloom and Kirk McKusick and the many others who have reviewed this paper giving considerable advice.

This work was sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871 monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency, of the US Government, or of Digital Equipment Corporation.

Update for the 4.9 release: the alpha-test group was extremely helpful in furnishing improvements, finding and repairing bugs, and being patient. I would like to express special thanks to Brian Reid for funding this work. Robert Elz, Alan Barrett, Paul Albitz, Bryan Beecher, Andrew Partan, Andy Cherenson, Tom Limoncelli, Berthold Paffrath, Fuat Baran, Anant Kumar, Art Harkin, Win Treese, Don Lewis, Christophe Wolfhugel, and a cast of dozens all helped out above and beyond the call of duty. Special thanks to Phil Almquist, who got the project started and contributed a lot of the code and fixed several of the worst bugs. **[** *Paul Vixie, DECWRL and DECNSL, April '93* **]**.

REFERENCES

[Birrell] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M.D., ‘‘Grapevine: An Exercise in Distributed Computing.’’ In *Comm. A.C.M. 25,* 4:260-274 April 1982.

[RFC819] Su, Z. Postel, J., ‘‘The Domain Naming Convention for Internet User Applications.’’ *Internet Request For Comment 819* Network Information Center, SRI International, Menlo Park, California. August 1982.

[RFC974] Partridge, C., ‘‘Mail Routing and The Domain System.’’ *Internet Request For Comment 974* Network Information Center, SRI International, Menlo Park, California. February 1986.

[RFC1032] Stahl, M., ‘‘Domain Administrators Guide’’ *Internet Request For Comment 1032* Network Information Center, SRI International, Menlo Park, California. November 1987.

[RFC1033] Lottor, M., ‘‘Domain Administrators Guide’’ *Internet Request For Comment 1033* Network Information Center, SRI International, Menlo Park, California. November 1987.

[RFC1034] Mockapetris, P., ‘‘Domain Names - Concept and Facilities.’’ *Internet Request For Comment 1034* Network Information Center, SRI International, Menlo Park, California. November 1987.

[RFC1035] Mockapetris, P., ‘‘Domain Names - Implementation and Specification.’’ *Internet Request For Comment 1035* Network Information Center, SRI International, Menlo Park, California. November 1987.

[RFC1101] Mockapetris, P., ‘‘DNS Encoding of Network Names and Other Types.’’ *Internet Request For Comment 1101* Network Information Center, SRI International, Menlo Park, California. April 1989.

[RFC1123] R. Braden, Editor, ‘‘Requirements for Internet Hosts -- Application and Support’’ *Internet Request For Comment 1123* Network Information Center, SRI International, Menlo Park, California. October 1989.

[RFC1183] Everhart, C., Mamakos, L., Ullmann, R., and Mockapetris, P., ‘‘New DNS RR Definitions’’ *Internet Request For Comment 1183* Network Information Center, SRI International, Menlo Park, California. October 1990.

[Terry] Terry, D. B., Painter, M., Riggle, D. W., and Zhou, S., *The Berkeley Internet Name Domain Server.* Proceedings USENIX Summer Conference, Salt Lake City, Utah. June 1984, pages 23-31.

[Zhou] Zhou, S., *The Design and Implementation of the Berkeley Internet Name Domain (BIND) Servers.* UCB/CSD 84/177. University of California, Berkeley, Computer Science Division. May 1984.

[Mockapetris] Mockapetris, P., Dunlap, K, *Development of the Domain Name System* ACM Computer Communications Review 18, 4:123-133. Proceedings ACM SIGCOMM ’88 Symposium, August 1988.

[Liu] Liu, C., Albitz, P., *DNS and BIND* O’Reilly & Associates, Sebastopol, CA, 502 pages, ISBN 0-937175-82-X 1992