**Part 6** Administering DNS

**This part describes the *Domain Name System (DNS)* and how to administer it.**

CHAPTER 1

# Introduction to DNS

---

This chapter describes the structure and provides an overview of the Domain Name System (DNS).

See *Solaris Naming Setup and Configuration Guide* for information on initially setting up and configuring DNS service.

---

**Note −** One of the most common, and important, uses of DNS is connecting your network to the global Internet. In order to connect to the Internet, your network IP address must be registered with whomever is administering your parent domain. Who that administrator is varies according to your geographic location and type of parent domain.

## DNS Basics

The Domain Name System (DNS) is an application – layer protocol that is part of the standard TCP/IP protocol suite. This protocol implements the DNS name service, which is the name service used on the Internet.

This section introduces the basic DNS concepts. It assumes that you have some familiarity with network administration, particularly TCP/IP, and some exposure to other name services, such as NIS+ and NIS.

Refer to *Solaris Naming Setup and Configuration Guide* for information regarding initial setup and configuration of DNS.

---

**Note −** DNS, NIS+, NIS, and FNS provide similar functionality and sometimes use the same terms to define different entities. Thus, this chapter takes care to define terms like domain and name server according to their DNS functionality, a very different functionality than NIS+ and NIS domains and servers.

---

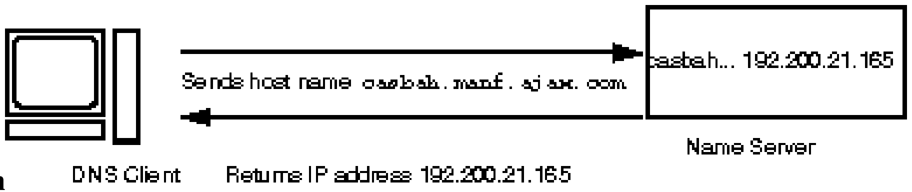
## Name−to−Address Resolution

Though it supports the complex, world−wide hierarchy of computers on the Internet, the basic function of DNS is actually very simple: providing *name−to−address resolution* for TCP/IP−based networks. Name−to−address resolution, also referred to as "mapping," is the process of finding the IP address of a computer in a database by using its host name as an index.

Name−to−address mapping occurs when a program running on your local machine needs to contact a remote computer. The program most likely will know the host name of the remote computer but may not know how to locate it, particularly if the remote machine is in another company, miles from your site. To get the remote machine’s address, the program requests assistance from the DNS software running on your local machine, which is considered a *DNS client*.

Your machine sends a request to a *DNS name server*, which maintains the distributed DNS database. The files in the DNS database bear little resemblance to the NIS+ Host Table or even the local /etc/hosts file, though they maintain similar information: the host names, IP addresses, and other information about a particular group of computers. The name server uses the host name your machine sent as part of its request to find or "resolve" the IP address of the remote machine. It then returns this IP address to your local machine IF the host name is in its DNS database.

@ *1−1* shows name−to−address mapping as it occurs between a DNS client and a name server, probably on the client’s local network.


casbah... 192.200.21.165
Sends host name casbah.manf.ajax.com
Name Server
DNS Client
Returns IP address 192.200.21.165

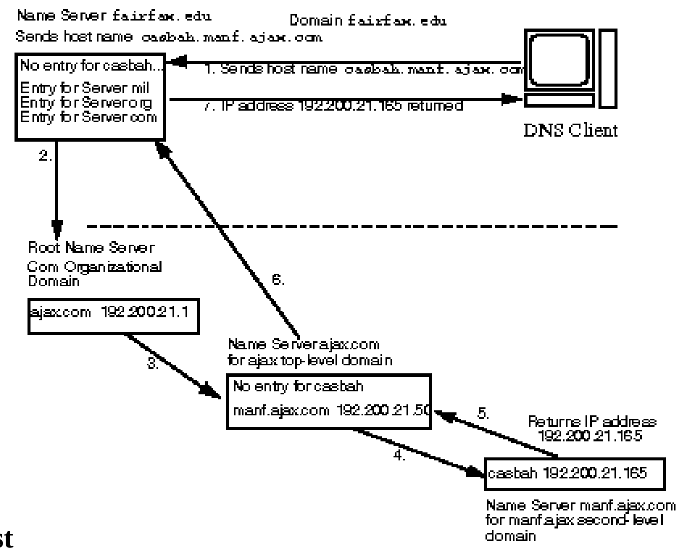
**Name to Address Resolution**

If the host name is not in that name server's DNS database, this indicates that the machine is outside of its authority, or, to use DNS terminology, outside the *local administrative domain*. Thus, each name server is spoken of as being "authoritative" for its local administrative domain.

Fortunately, the local name server maintains a list of host names and IP addresses of *root domain name servers*, to which it will forward the request from your machine. These root name servers are authoritative for huge organizational domains, as explained fully in *DNS Hierarchy and the Internet @ 1−3*. These hierarchies resemble UNIX file systems, in that they are organized into an upside−down tree structure.

Each root name server maintains the host names and IP address of top level domain name servers for a company, a university, or other large organizations. The root name server sends your request to the top−level name servers that it knows about. If one of these servers has the IP address for the host you requested, it will return the information to your machine. If the top−level servers do not know about the host you requested, they pass the request to second−level name servers for which they maintain information. Your request is then passed on down through the vast organizational tree. Eventually, a name server that has information about your requested host in its database will return the IP address back to your machine.

*@ 1−1* shows name−to−address resolution outside the local domain.


Name Server fairfax.edu
Sends host name casbah.manf.ajax.com
Domain fairfax.edu
No entry for casbah...
Entry for Server mil
Entry for Server org
Entry for Server com
1. Sends host name casbah.manf.ajax.com
7. IP address 192.200.21.165 returned
DNS Client
2.
Root Name Server
Com Organizational Domain
ajax.com 192.200.21.1
6.
3.
Name Server ajax.com
for ajax top-level domain
No entry for casbah
manf.ajax.com 192.200.21.50
5.
Returns IP address 192.200.21.165
4.
casbah 192.200.21.165
Name Server manf.ajax.com
for manf.ajax second-level domain

**Name to Address Resolution for a Remote Host**

## DNS Administrative Domains

From a DNS perspective, an *administrative domain* is a group of machines that are administered as a unit. Information about this domain is maintained by at least two name servers; they are "authoritative" for the domain. The DNS domain is a purely logical grouping of machines. It could correspond to a physical grouping of machines, such as all machines attached to the Ethernet in a small business. But a local DNS domain just as likely could include all machines on a vast university network that belong to the computer science department or to university administration.

For example, suppose the Ajax company has two sites, one in San Francisco and one in Seattle. The Retail.Sales.Ajax.com. domain might be in Seattle and the Wholesale.Sales.Ajax.com. domain might be in San Francisco. One part of the Sales.Ajax.com. domain would be in one city, the other part in the second city.

Each administrative domain must have its own unique subdomain name. Moreover, if you want your network to participate in the Internet, the network must be part of a registered administrative domain. The section *Joining the Internet @ 1−1* has full details about domain names and domain registration.

## `in.named` and DNS Name Servers

As mentioned previously, name servers in an administrative domain maintain the DNS database. They also run the `in.named` daemon, which implements DNS services, most significantly, name−to−address mapping. `in.named` is a public domain TCP/IP program and included with the Solaris 2.6 Solaris 2.6 releaseenvironment.

---

**Note −** The `in.named` daemon is also called the Berkeley Internet Name Domain service, or BIND, because it was developed at the University of California at Berkeley.

---

There are three types of DNS name servers:

- Primary server
- Secondary server
- Cache−only server

Each domain must have one primary server and should have at least one secondary server to provide backup. *Zones @ 1−3* explains primary and secondary servers in detail.

## DNS Clients and the Resolver

To be a DNS client, a machine must run the *resolver*. The resolver is neither a daemon nor a single program; rather, it is a set of dynamic library routines used by applications that need to know machine names. The resolver's function is to resolve users' queries. To do that, it queries a name server, which then returns either the requested information or a referral to another server. Once the resolver is configured, a machine can request DNS service from a name server.

When a machine's /etc/nsswitch.conf file specifies hosts: dns (or any other variant that includes dns in the hosts line), the resolver libraries are automatically used. If the nsswitch.conf file specifies some other name service before dns, that name service is consulted first for host information and only if that name service does not find the host in question are the resolver libraries used.

For example, if the hosts line in the nsswitch.conf file specifies hosts: nisplus dns, the NIS+ name service will first be searched for host information. If the information is not found in NIS+, then the DNS resolver is used. Since name services such as NIS+ and NIS only contain information about hosts in their own network, the effect of a hosts:nisplus dns line in a switch file is to specify the use of NIS+ for local host information and DNS for information on remote hosts out on the Internet.

There are two kinds of DNS clients:

- *Client−only*. A client−only DNS client does not run `in.named`; instead, it consults the resolver. The resolver knows about a list of name servers for the domain, to which queries are then directed.
- *Client−server*. A client−server uses the services provided by `in.named` to resolve queries forwarded to it by client−machine resolvers.

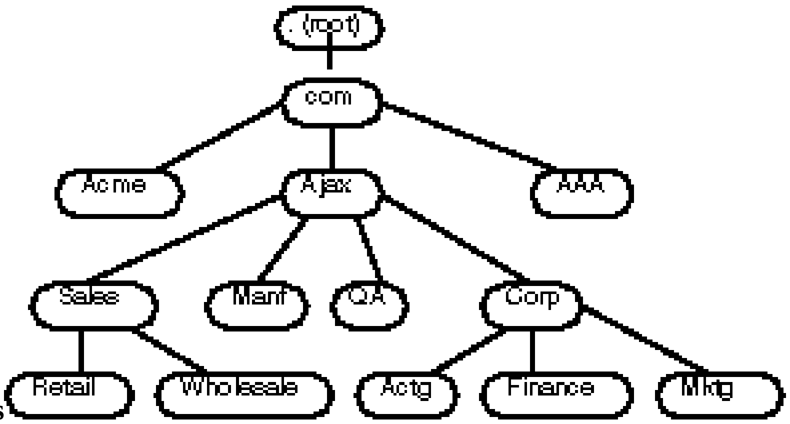
The Solaris 2.6 release environment includes the dynamic library routines that make up the resolver. *Solaris Naming Setup and Configuration Guide,* contains instructions for setting up a host as a DNS client.

## Introducing the DNS Namespace

The entire collection of DNS administrative domains throughout the world are organized in a hierarchy called the *DNS namespace*. This section shows how the namespace organization affects both local domains and the Internet.

## DNS Namespace Hierarchy

Like the UNIX file system, DNS domains are organized as a set of descending branches similar to the roots of a tree. Each branch is a domain, each subbranch is a *subdomain.* The terms *domain* and *subdomain* are relative. A given domain is a subdomain relative to those domains above it in the hierarchy, and a parent domain to the subdomains below it.



**Domains and Subdomains**

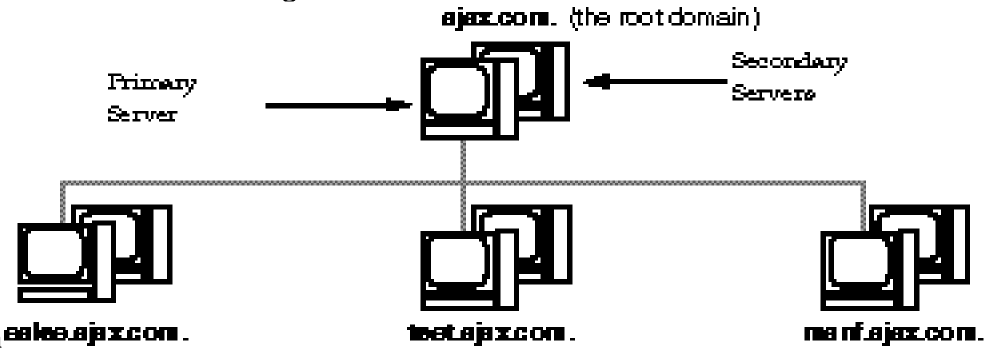
For example, in    @  *1−1*, com is a parent domain to the Acme, Ajax, and AAA domains. Or you could just as easily say that those are subdomains relative to the com domain. In its turn, the Ajax domain is a parent to four subdomains (Sales, Manf, QA, and Corp).

A domain contains one parent (or top) domain plus the associated subdomains if any. Domains are named up the tree starting with the lowest (deepest) subdomain and ending with the root domain. For example, Mktg.Corp.Ajax.Com. from    @  *1−1*.

## DNS Hierarchy in a Local Domain

If your company is large enough, it may support a number of domains,organized into a local namespace. @ *1−1* shows a domain hierarchy that might be in place in a single company. The top−level, or "root" domain for the organization is ajax.com, which has three sub−domains, sales.ajax.com, test.ajax.com, and manf.ajax.com.

**Hierarchy of DNS Domains in a Single Organization**

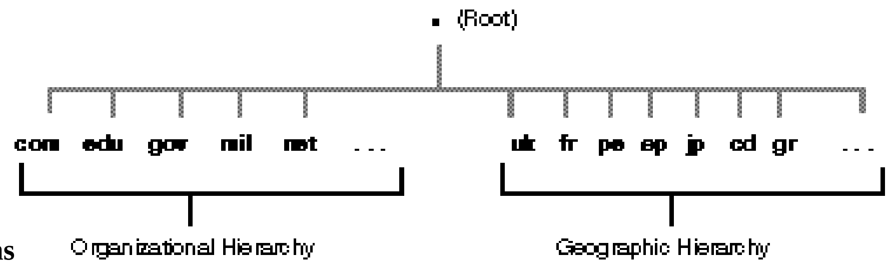


DNS clients request service only from the servers that support their domain. If the domain's server does not have the information the client needs, it forwards the request to its parent server, which is the server in the next−higher domain in the hierarchy. If the request reaches the top−level server, the top−level server determines whether the domain is valid. If it is *not* valid, the server returns a "not found" type message to the client. If the domain is valid, the server routes the request down to the server that supports that domain.

## DNS Hierarchy and the Internet

The domain hierarchy shown in @ *1−1* is, conceptually, a "leaf" of the huge DNS namespace supported on the global Internet.

The DNS namespace for the Internet is organized hierarchically as shown @ *1−1*. It consists of the root directory, represented as a dot (.) and two top level domain hierarchies, one organizational and one geographical. Note that the com domain introduced in @ *1−1*is one of a number of top−level organizational domains in existence on the Internet.



**Hierarchy of Internet Domains**

At the present time, the organizational hierarchy divides its namespace into the top−level domains listed shown in *1−1*. It is probable that additional top−level organizational domains will be added in the future.

**1−1 Internet Organizational Domains**

| Domain | Purpose |
|---|---|
| com | Commercial organizations |

| | |
|---|---|
| edu | Educational institutions |
| gov | Government institutions |
| mil | Military groups |
| net | Major network support centers |
| org | Nonprofit organizations and others |
| int | International organizations |

The geographic hierarchy assigns each country in the world a two− or three−digit identifier and provides official names for the geographic regions within each country. For example, domains in Britain are subdomains of the uk top−level domain, Japanese domains are subdomains of jp, and so on.

## Joining the Internet

The Internet root domain, top−level domains (organizational and geographical) are maintained by the various Internet governing bodies. People with networks of any size can "join" the Internet by registering their domain name in either the organizational or the geographical hierarchy.

Every DNS domain must have a domain name. If your site wants to use DNS for name service *without* connecting to the Internet, you can use any name your organization wants for its your domains and subdomains, if applicable. However, if your site plans wants to join the Internet, it *must* register its domain name with the Internet governing bodies.

To join the Internet, you have to:

- Register your DNS domain name with the an appropriate Internet governing body.
- Obtain a network IP address from that governing body.

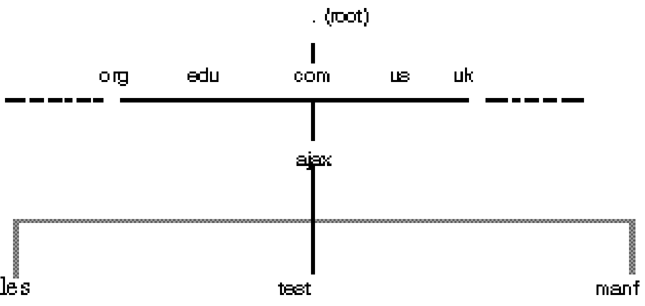
There are two ways to accomplish this:

- You can communicate directly with the appropriate Internet governing body or their agent. In the United States, InterNIC is the company that currently handles network address and domain registration matters.
- You can contract with an Internet Service Provider (ISP) to assist you. ISPs provide a wide range of services from consulting to actually hosting your Internet presence.

## Domain Names

Domain names indicate a domain’s position in the overall DNS namespace, much as path names indicate a file's position in the UNIX file system. After your local domain is registered, its name is prepended to the name of the Internet hierarchy to which it belongs. For example, the ajax domain shown in @ *1−1*has been registered as part of the Internet com hierarchy. Therefore, its Internet domain name becomes ajax.com.

@ *1−1* shows the position of the ajax.com domain in the DNS namespace on the Internet.



**Ajax Domain's Position in the DNS Namespace**

The ajax.com subdomains now have the following names.

```
sales.ajax.com
 test.ajax.com
 manf.ajax.com
```

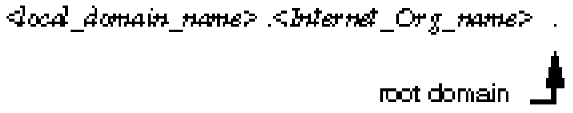
DNS does not require domain names to be capitalized, though they may be. Here are some examples of machines and domain names:

```
Boss.manf.ajax.com
 quota.Sales.ajax.com
```

The Internet organization regulates administration of its domains by granting each domain authority over the names of its hosts and by expecting each domain to delegate authority to the levels below it. Thus, the com domain has authority over the names of the hosts in its domain. It also authorizes the formation of the Ajax.com domain and delegates authority over the names in that domain. The Ajax.com domain, in turn, assigns names to the hosts in its domain and approves the formation of the Sales.Ajax.com, Test.Ajax.com, and Manf.Ajax.com domains.

## Fully−Qualified Domain Names

A domain name is said to be *fully−qualified* when it includes the names of every DNS domain from the local domain on up to ".", the DNS root domain. Conceptually, the fully−qualified domain name indicates the path to the root, as does the absolute path name of a UNIX file. However, fully−qualified domain names are read from lowest, on the left, to highest, on the right. Therefore, a fully−qualified domain name



has the syntax:

The fully qualified domain names for the ajax domain and its subdomains are:

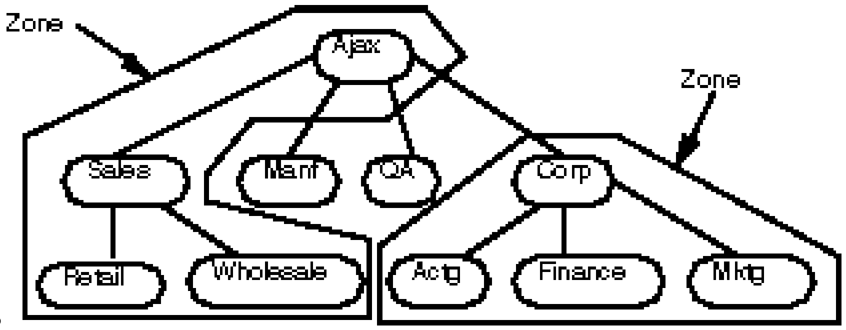
```
ajax.com.
sales.ajax.com
test.ajax.com.
manf.ajax.com
```

Note the dot at the furthest right position of the name.

## Zones

DNS service for a domain is managed on the set of name servers first introduced *in.named and DNS Name Servers @ 1−3*. Name servers can manage a single domain, or multiple domains, or domains and some or all of their corresponding subdomains. The part of the namespace that a given name server controls is called a *zone*; thus, the name server is said to be authoritative for the zone. If you are responsible for a particular name server, you may be given the title zone administrator.

The data in a name server's database are called *zone files*. One type of zone file stores IP addresses and host names. When someone attempts to connect to a remote host using a host name by a utility like `ftp` or `telnet`, DNS performs name−to−address mapping, by looking up the host name in the zone file and converting it into its IP address.



**Domains and Zones**

For example, the Ajax domain shown in *@ 1−1* contains a top domain (Ajax), four subdomains, and five sub−subdomains. It is divided into four zones shown by the thick lines. Thus, the Ajax name server administers a zone composed of the Ajax, Sales, Retail, and Wholesale domains. The Manf and QA domains are zones unto themselves served by their own name servers, and the Corp name server manages a zone composed of the Corp, Actg, Finance, and Mktg domains.

## Reverse Mapping

The DNS database also include zone files that use the IP address as a key to find the host name of the machine, enabling IP address to host name resolution. This process is called *reverse resolution* or more commonly, reverse mapping. Reverse mapping is used primarily to verify the identity of the machine that sent a message or to authorize remote operations on a local host.

## The in−addr.arpa Domain

The in−addr.arpa domain is a conceptual part of the DNS namespace that uses IP addresses for its leaves, rather than domain names. It is the part of your zone that enables address to name mapping.

Just as DNS domain names are read with the lowest level subdomain occupying the furthest left position and the root at the far right, in−addr.arpa domain IP addresses are read from lowest level to the root. Thus, the IP addresses are read backward. For example, suppose a host has the IP address 192.200.21.165. In the in−addr.arpa zone files, its address is listed as 165.21.200.192.in−addr.arpa. with the dot at the end indicating the root of the in−addr.arpa domain.

## DNS Servers

DNS servers perform one or more functions:

- *Zone Master Servers.*A master name server maintains all the data corresponding to the zone, making it the authority for that zone. Master servers are commonly called *authoritative* name servers. (See *Master Servers @ 1−1.*)

  There are two types of master server:

  - *Zone primary master server.* Each zone has one server that is designated as the *primary* master server for that zone. (See *Primary Master Server @ 1−1.*)
  - *Zone secondary master server.*A zone may have one or more *secondary* master servers. Secondary master servers obtain their DNS data from the zone's primary master server. (See *Primary Master Server @ 1−1.*)
- *Cache−only Server.* All servers are caching servers in the sense that they maintain a cache of DNS data. A cache−only server is a server that is not a master server for any zone other than the in−addr.arpa. domain. (See *Caching and Cache−only Servers @ 1−2.*)
- *Root Domain servers.* A root domain server is the authoritative server for the top of your DNS domain hierarchy. If your network is connected to the Internet, your root domain servers are out on the Internet itself. If your network is not connected to the Internet, you must set up your own root domain server. (See *Root Domain Name Server @ 1−3.*)

These different server functions can be performed by the same machine.For example, a machine can be a primary master server for one zone and a secondary master server for another zone. When this manual refers to a primary or secondary or cache−only server, it is not referring to a particular machine, but the role that machine plays for a given zone.

## Master Servers

The master name servers maintain all the data corresponding to the zone, making them the authority for that zone. These are commonly called *authoritative* name servers. The data corresponding to any given zone should be available on at least two authoritative servers. You should designate one name server as the *primary* master server and at least one more as a *secondary* master server, to act as a backup if the primary is unavailable or overloaded.

A server may function as a master for multiple zones: as a primary for some zones, and as a secondary for others.

## Primary Master Server

The *primary* master server is the DNS name server that loads the master copy of its data from disk when it starts in.named. A zone's primary master server is where you make changes for the zone. The primary master is the source for DNS information regarding its zone. The primary server may also delegate authority to secondary servers in its zone as well as to servers outside its zone.

## Secondary Master Server

A *secondary* master server maintains a copy of the data for the zone. The primary server sends its data and delegates authority to the secondary server. Clients can query a secondary server for DNS information.By using secondary servers, you can improve response time and reduce network overhead by spreading the load over multiple machines. Secondary servers also provide redundancy in case the primary server is not available.

When the secondary server starts in.named, it requests all the data for the given zone from the primary. The secondary server then periodically checks with the primary to see if it needs to update its database. The process of sending the most recent zone database from the primary to the secondary is called a *zone transfer*. Thus, you do not modify data files on a secondary server, you modify the data files on the zone's primary server and the secondary servers update their files from the primary.

## Caching and Cache−only Servers

All name servers are *caching servers*. This means that the name server caches received information until the data expires. The expiration process is regulated by the time−to−live (TTL) field that may be attached to the data.

Additionally, you can set up a *cache−only server* that is not authoritative for any zone. A cache−only server is a server that is not a master server for any zone other than the in−addr.arpa. domain. A cache−only server handles the same kind of queries from clients that authoritative name servers perform. But the cache−only server does not maintain any authoritative data itself.

A cache−only server requires less memory than an authoritative server, but cannot function by itself if no primary or secondary servers are available.

## Root Domain Name Server

A DNS name space must have one ore more *root domain name servers* that are authoritative for the root domain.

- If your network is connected to the Internet, your root domain server exists at the root domain Internet site and all you have to do is provide that site's Internet IP addresses in your cache file as explained in *Internet Root Domain Server @ 1−1*.
- If your network is not connected to the Internet, you must set up primary and secondary name servers

in the root−level domain on your local network as explained in *Non−Internet Root Domain Server @ 1−2*. This is so that all domains in your network have a consistent authoritative server to which to refer; otherwise, machines may not be able to resolve queries.

The information that identifies the root domain name servers is stored in a cache file. This manual and most Solaris sites call this file named.ca. (Other common names for this file are: root.cache, named.root, or db.cache.) Each server's boot file contains a record identifying the file that holds the root domain name server information.

## Internet Root Domain Server

If your site is connected to the Internet, your DNS name server's boot files must point to a common cache file (usually called named.ca) that identifies the root domain name servers. A template for this file may be obtained from InterNIC registration services via:

- Anonymous FTP. The FTP site is: ftp.rs.internic.net. The file name is: /domain/named.root.
- Gopher. The Gopher site is: rs.internic.net. The file is: named.root which can be found under the InterNIC Registration Services menu, InterNIC Registration Archives submenu.

If you are naming your DNS files according to the conventions in this manual, you need to move this file to /var/named/named.ca.

## Non−Internet Root Domain Server

If your site is not connected to the Internet, you must set up one or more of your servers to perform as root domain name servers. The boot files of all DNS name servers on your network must point to a common cache file (usually called named.ca) that identifies the root domain name servers. You then create a cache file that identifies your root name servers.

Since a single machine can be the primary domain name server for more than one machine, the easiest way to create a root domain name server is to have the server for your highest level domain also be the server for the logical "." domain.

For example, suppose you have given your network the domain name solo. The DNS master name server is dnsmaster.solo.(with a trailing dot). In this case, you would make dnsmaster the root master server for the "." domain.

If your network has more than one top−level domain, the root domain server name should be the primary name server for all top−level domains. For example, if your network is divided into two separate, non−hierarchal domains named solo and private, the same server must be root master server for both of them. Following the example above that would mean that dnsmaster.solo. is root domain master for both the solo and the private domains.

## How DNS Affects Mail Delivery

DNS provides two principal services, it performs name to address mapping (and also maps addresses to host names), as discussed in *Name−to−Address Resolution @ 1−1*. It also helps mail delivery agents, such as `sendmail` and POP, deliver mail along the Internet.

To deliver mail across the Internet, DNS uses *mail exchange records* (MX records). Most organizations don't allow direct delivery of mail that comes across the Internet for hosts within the organization. Instead, they use a central mail host (or a set of mail hosts) to intercept incoming mail messages and route them to their recipients.

The mail exchange record identifies the mail host that services each machine in a domain. Therefore, a mail exchange record lists the DNS domain names of remote organizations and either the IP address or the host name of its corresponding mail host.

## DNS Boot and Data Files

In addition to the `in.named` daemon, DNS on a name server consists of a boot file called named.boot, a resolver file named resolv.conf, and four types of zone data files.

## Names of DNS Data Files

So long as you are internally consistent, you can name the zone data files anything you want. This flexibility may lead to some confusion when working at different sites or referring to different DNS manuals and books.

For example, the file names used in Sun manuals and at most many Solaris sites vary from those used in the book *DNS and BIND* by Albitz and Liu, O'Reilly & Associates, 1992, and both of those nomenclatures have some differences from that used in the public−domain *Name Server Operations Guide for BIND*, University of California.

In addition, this manual and other DNS documentation uses generic names that identify a file's main purpose, and specific example names for that file in code record samples. For example, *Solaris Naming* manuals use the generic name hosts when describing the function and role of that file, and the example names db.doc and db.sales.doc in code samples.

For reference purposes, *1−1* compares BIND file names from these three sources:

**1−1 BIND File Name Examples**

| Solaris Names | O'Reilly Names or other names | U.C. Berkeley Names | Content and Purpose of File |
|---|---|---|---|
| /etc/named.boot | /etc/named.boot | /etc/named.boot | The boot file specifies the type of server it is running on and the zones over which it has control. It contains a list of domain names and the names of the data files. |
| /etc/resolv.conf | /etc/resolv.conf | /etc/resolv.conf | This file resides on every DNS client (including DNS servers) and designates the servers which the client queries for DNS |

| | | | |
|---|---|---|---|
| | | | information. |
| named.ca | db.cache<br>db.root | root.cache | This file establishes the names of root servers and lists their addresses. |
| Generic: hosts<br>Examples: db.doc<br>db.sales | Generic: db.domain<br>Examples: db.movie<br>db.fx | Generic: hosts<br>Example: ucbhosts | This file contains all the data about the machines in the local zone that the server serves. |
| Generic: hosts.rev<br>Examples:<br>doc.rev | Generic: db.ADDR<br>Examples:<br>db.192.249.249<br>db.192.249.253 | hosts.rev | This file specifies a zone in the in−addr.arpa. domain, a special domain that allows reverse (address−to−name) mapping. |
| named.local | Generic: db.cache<br>Example: db.127.0.0 | named.local | This file specifies the address for the local loopback interface, or localhost |
| $INCLUDE files | $INCLUDE files | $INCLUDE files | Any file identified by an $INCLUDE statement in a data file. |

## The named.boot File

The boot file (/etc/named.boot) establishes the server as a primary, secondary, or cache−only name server. It also specifies the zones over which the server has authority and which data files it should read to get its initial data.

The boot file is read by `in.named` when the daemon is started by the server's start up script, /etc/init.d/inetsvc. The boot file directs `in.named` either to other servers or to local data files for a specified domain.)

## The named.ca File

The named.ca file establishes the names of root servers and lists their addresses. If your network is connected to the Internet, named.ca lists the Internet name servers; otherwise, it lists the root domain name servers for your local network. The `in.named` daemon cycles through the list of servers until it contacts one of them. It then obtains from that server the current list of root servers, which it uses to update named.ca.

## The hosts File

The hosts file contains all the data about the machines in the local zone. The name of this file is specified in the boot file. To avoid confusion with /etc/hosts, name the file something other than hosts, for example, you could name these files using the pattern db.*domain*. Using that nomenclature, the host files for the doc.com and sales.doc.com domains might be db.doc and db.sales.

## The hosts.rev File

The hosts.rev file specifies a zone in the in−addr.arpa. domain, the special domain that allows reverse (address−to−name) mapping. The name of this file is specified in the boot file.

## The named.local File

The named.local file specifies the address for the local loopback interface, or localhost, with the network address 127.0.0.1. The name of this file is specified in the boot file. Like other files, you can give it a name other than the name used in this manual.

## $INCLUDE Files

An include file is any file named in an $INCLUDE statement in a DNS data file. $INCLUDE files can be used to separate different types of data into multiple files for your convenience.

For example, suppose a data file contained following line:

```
$INCLUDE /etc/named/data/mailboxes
```

This line causes the /etc/named/data/mailboxes file to be loaded at that point. In this instance, /etc/named/data/mailboxes is an $INCLUDE file. Use of $INCLUDE files is optional. You can use as many as you wish, or none at all.

## Data File Resource Record Format

All the data files used by the DNS daemon in.named are written in standard resource record format. Each DNS data file must contain certain resource records. This section describes the DNS data files and the resource records each file should contain.

## Standard Resource Record Format

In the standard resource record format, each line of a data file is called a *resource record* (RR), which contains the following fields separated by white space:

```
namettl   class  record−type    record−specific−data
```

The order of the fields is always the same; however, the first two are optional (as indicated by the brackets), and the contents of the last vary according to the *record−type* field.

## The *name* Field

The first field is the name of the domain that applies to the record. If this field is left blank in a given RR, it defaults to the name of the previous RR.

A domain name in a zone file can be either a fully qualified name, terminated with a dot, or a relative name, in which case the current domain is appended to it.

## The *ttl* Field

The second field is an optional time−to−live field. This specifies how long (in seconds) this data will be cached in the database before it is disregarded and new information is requested from a server. By leaving this field blank, the *ttl* defaults to the minimum time specified in the Start−Of−Authority (SOA) resource record.

If the *ttl* value is set too low, the server will incur a lot of repeat requests for data refreshment; if, on the other hand, the *ttl* value is set too high, changes in the information will not be timely distributed.

Most *ttl* values should be initially set to between a day (86400) and a week (604800). Then, depending on the frequency of actual change of the information, you can change the appropriate *ttl* values to reflect that frequency. Also, if you have some *ttl* values that have very high numbers because you know they relate to data that rarely changes. When you know that the data is now about to change, reset the *ttl* to a low value (3600 to 86400) until the change takes place. Then change it back to the original high value.

All RR's with the same name, class, and type should have the same *ttl* value.

## The *class* Field

The third field is the record *class*. Only one *class* is currently in use: IN for the TCP/IP protocol family.

## The *record−type* Field

The fourth field states the resource record *type*. There are many types of RR's; the most commonly used types are discussed in *Resource Record Types @ 1−4*.

## The *record−specific−data* Field

The contents of the *record−specific−data* field depend on the type of the particular resource record.

Although case is preserved in names and data fields when loaded into the name server, all comparisons and lookups in the name server database are case insensitive. However, this situation may change in the future; thus, you should be consistent in your use of lower and uppercase.

## Special Resource Record Characters

The following characters have special meanings:

**1−1  Special Resource Record Characters**

| Character | Definition |
|---|---|
| . | 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, you can use \\. to place a dot character in a label. |
| \*DDD* | Where each *D* is a digit, this 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. |
| () | Use parentheses to group data that crosses a line. In effect, line terminations are not recognized within parentheses. |
| ; | A semicolon starts a comment; the remainder of the line is ignored. |
| * | An asterisk signifies a wildcard. |

Most resource records have the current origin appended to names if they are not terminated by a dot (.) This is useful for appending the current domain name to the data, such as machine names, but may cause problems when you do not want this to happen. You should use a fully qualified name ending in a period if the name is not in the domain for which you are creating the data file.

## Control Entries

The only lines that do not conform to the standard RR format in a data file are control−entry lines. There are two kinds of control entries: $INCLUDE and $ORIGIN.

## $INCLUDE

An include line begins with $INCLUDE in column 1, and is followed by a file name (known as the $INCLUDE file). This feature is particularly useful for separating different types of data into multiple files as in this example:

```
$INCLUDE /etc/named/data/mailboxes
```

The line is interpreted as a request to load the /etc/named/data/mailboxes file at that point. 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 zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

Use of $INCLUDE statements and files is optional. You can use as many as you wish, or none at all.

## $ORIGIN

The $ORIGIN command is a way of changing the origin in a data file. The line starts in column 1, and is followed by a domain name. It resets the current origin for relative domain names (for example, not fully qualified names) to the stated name. This is useful for putting more than one domain in a data file.

**Note −** You cannot use $ORIGIN for putting more than one zone in a data file.

Use of $ORIGIN commands in a data file is optional. If there is no $ORIGIN statement the default origin for DNS data files is the domain named in the second field of the primary or secondary line of the named.boot file.

## Resource Record Types

The most commonly used types of resource records are listed in *1−1*. They are usually entered in the order shown in *1−1*, but that is not a requirement.

**1−1 Commonly Used Resource Record Types**

| Type | Description |
|---|---|
| SOA | Start of authority |
| NS | Name server |
| A | Internet address (name to address) |
| PTR | Pointer (address to name) |
| CNAME | Canonical name (nickname) |

| | |
|---|---|
| TXT | Text information |
| WKS | Well−known services |
| HINFO | Host information |
| MX | Mail exchanger |

## SOA — Start of Authority

*SOA Record Format @ 1−1* shows the syntax of a start−of−authority (SOA) resource record.

**SOA Record Format**

```
name class SOA origin person−in−charge ( serial number
  refresh
 retry
 expire
 ttl)
```

The Start−Of−Authority record designates the start of a zone. The zone ends at the next SOA record. The SOA record fields are described below.

### name

This field indicates the name of the zone. Note that the zone name must end with a trailing dot. For example: doc.com. is correct, while doc.com is wrong.

### class

This field is the address class. For example, IN for Internet (the most commonly used class).

### SOA

This field is the type of this resource record.

### origin

This field is the name of the host where this data file resides. Note that this host name must end in a trailing dot. For example, dnsmaster.doc.com. is correct, but dnsmaster.doc.com is wrong.

**person−in−charge**

This field is the email address of the person responsible for the name server. For example, kjd.nismaster.doc.com. Again, this name must end with a trailing dot.

**serial**

This field is the version number of this data file. You must increment this number whenever you make a change to the data: secondary servers use the serial field to detect whether the data file has been changed since the last time they copied the file from the master server.

**refresh**

This field indicates how often, in seconds, a secondary name server should check with the primary name server to see if an update is needed. For example, 7200 indicates a period of two hours.

**retry**

This field indicates how long, in seconds, a secondary server is to retry after a failure to check for a refresh.

**expire**

This field 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.

**ttl**

This field is the default number of seconds to be used for the time−to−live field on resource records that don’t have a *ttl* specified elsewhere.

There should only be one SOA record per zone. *Sample SOA Resource Record @ 1−1*is a sample SOA resource record.

**Sample SOA Resource Record**

```
;name class   SOA  origin     person−in−charge
```

```
doc.com. IN  SOA dnsmaster.doc.com. root.nismaster.doc.com. (
       101   ;Serial
       7200  ;Refresh
       3600  ;Retry
       432000  ;Expire
       86400)  ;Minimum    )
```

## NS −Name Server

*NS Record Format @  1−1* shows the syntax of a name−server (NS) resource record:

**NS Record Format**
`domainname [optional TTL] class NS name−server−name`

The name−server record lists by name a server responsible for a given domain. The *name* field lists the domain that is serviced by the listed name server. If no *name* field is listed, then it defaults to the last name listed. One NS record should exist for each primary and secondary master server for the domain. *Sample NS Resource Record @  1−2*is a sample NS resource record.

**Sample NS Resource Record**
```
;domainname    [TTL]   class NS nameserver
doc.com        90000     IN NS  sirius.doc.com.
```

## A −Address

*Address Record Format @  1−1* shows the syntax of an address (A) resource record:

**Address Record Format**
` machinename [optional TTL] class A address`

The address (A) record lists the address for a given machine. The *name* field is the host name, and the *address* is the IP address. One A record should exist for each address of the machine (in other words, routers, or gateways require at least two entries, a separate entry including the IP address assigned to each network interface).

**Sample Address Record**
```
;machinename [TTL] class A address
sirius  IN  A 123.45.6.1
```

## HINFO −Host Information

*HINFO Record Format @ 1−1* shows the syntax of a host−information (HINFO) resource record:

**HINFO Record Format**

```
[optional name] [optional TTL]  class HINFO hardware OS
```

The host−information resource record (HINFO) contains host−specific data. It lists the hardware and operating system that are running at the listed host. If you want to include a space in the machine name or in the entry in the *hardware* field, you must surround the entry with quotes. The *name* field specifies the name of the host. If no name is specified, it defaults to the last `in.named` host. One HINFO record should exist for each host. *Sample HINFO Resource Record @ 1−2*is a sample HINFO resource record.

**Sample HINFO Resource Record**

```
;[name]   [TTL] class HINFO   hardware    OS
                IN    HINFO   Sparc−10    UNIX
```

---

**Caution −** Because the HINFO field provides information about the machines on your network, many sites consider it a security risk and no longer use it.

---

## WKS −Well−Known Services

*WKS Record Format @ 1−1* shows the syntax of a well−known services (WKS) resource record:

**WKS Record Format**

```
[Optional name] [TTL] class WKS address protocol−list−of−services
```

The Well−Known Services (WKS) record 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 the services database. Only one WKS record should exist per protocol per address. *Sample WKS Resource Record @ 1−2*is an example of a WKS resource record.

**Sample WKS Resource Record**

```
;[name] [TTL] class WKS address   protocol−list−of−services
altair  IN WKS 123.45.6.1  TCP ( smtp discard rpc
                                                 sftp uucp−path systat
daytime
       netstat qotd nntp doc.com )
```

---

**Caution −** The WKS record is optional. For security reasons, most sites no longer provide this information.

---

## CNAME −Canonical Name

*CNAME Record Format @ 1−1* shows the syntax of a canonical−name (CNAME) resource record.

**CNAME Record Format**

```
 nickname [optional TTL] class CNAME canonical−name
```

The Canonical−Name Resource record (CNAME) specifies a nickname or alias for a canonical name. A nickname should be unique. All other resource records should be associated with the canonical name and not with the nickname. Do not create a nickname and then use it in other resource records. Nicknames are particularly useful during a transition period, when a machine's name has changed but you want to permit people using the old name to reach the machine. Nicknames can also be used to identify machines that serve some specific purpose such as a mail server. *Sample CNAME Resource Record @ 1−2*is a sample CNAME resource record.

**Sample CNAME Resource Record**

```
;nickname      [TTL] class CNAME canonical−name
mailhost  IN CNAME antares.doc.com
```

## PTR −Pointer Record

*PTR Record Format @ 1−1* shows the syntax for a pointer (PTR) resource record.

**PTR Record Format**

```
 special−name [optional TTL]  class  PTR−real−name
```

A pointer record allows special names to point to some other location in the domain. In the example, PTR's are used mainly in the in−addr.arpa. records for the translation of an address (the special name) to a real name. When translating an address, if the domain is fully qualified only the machine identification number need be specified. PTR names should be unique to the zone. The PTR records *Sample PTR Resource Record @ 1−2*sets up reverse pointers for the special in−addr.arpa domain.

**Sample PTR Resource Record**

```
;special name   [TTL] class PTR−real−name
1   IN PTR sirius.doc.com.
```

## MX −Mail Exchanger

*MX Record Format @ 1−1* shows the syntax for a mail−exchanger (MX) resource record.

**MX Record Format**

*name* [optional *TTL*] *class* MX *preference−value mailer−exchanger*

The mail−exchanger resource records are used to specify a machine that knows how to deliver mail to a domain or specific machines in a domain. There may be more than one MX resource record for a given name. In *Sample MX Resource Record @ 1−2*, Seismo.CSS.GOV. (note the fully qualified domain name) is a mail gateway that knows how to deliver mail to Munnari.OZ.AU. Other machines on the network cannot deliver mail directly to Munnari. Seismo and Munnari may have a private connection or use a different transport medium. The *preference−value* field indicates the order a mailer should follow when there is more than one way to deliver mail to a single machine. The value 0 (zero) indicates the highest preference. If there is more than one MX resource record for the same name, records may or may not have the same *preference* value.

You can use names with the wildcard asterisk (*) 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. In *Sample MX Resource Record @ 1−2*, all mail to hosts in domain foo.com is routed through RELAY.CS.NET. You do this by creating a wildcard resource record, which states that the mail exchanger for *.foo.com is RELAY.CS.NET. The asterisk will match any host or subdomain of foo.com, but it will not match foo.com itself.

**Sample MX Resource Record**

```
;name   [TTL] class  MX  preference mailer−exchanger
Munnari.OZ.AU.  IN  MX  0 Seismo.CSS.GOV.
foo.com.  IN  MX  10 RELAY.CS.NET.
*.foo.com.  IN  MX  20 RELAY.CS.NET.
```

## Solaris DNS BIND Implementation

For your convenience, the Solaris 2.6 releasesupplies a compiled version of Berkeley Internet Name Domain (BIND) version 4.9.4, Patch−Level 1. In compiling this software, options and choices were made to meet the needs of the greatest number of sites. If this pre−compiled version of BIND does not meet your requirements, you can recompile your own version of BIND from the publicly available source code.

In compiling the BIND version supplied with the Solaris 2.6 release, the following choices were made:

- *RFC1535*. Not implemented since because doing so would remove implicit search lists.
- *Inverse Queries*. Enabled because SunOS 4.x `nslookup` will not work without them.
- *Bogus Name Logging.* Logging of bogus name servers is not implemented because it produces too many unimportant messages.
- *Default Domain Name*. If the DNS domain name is not set in /etc/resolv.conf, or via the LOCALDOMAIN environment variable, `libresolv` derives it from the NIS or NIS+ domain name provided that the /etc/nsswitch.conf file contains nisplus or nis as the first element in the hosts line.
- *Utility Scripts.* The BIND utility scripts are not included in this Solaris release.
- *Test Programs*. The BIND test programs `dig`, `dnsquery`, and `host` are not included in this Solaris release because their purpose is similar to that of `nslookup` and `nstest`.

CHAPTER 2

# Administering DNS

This chapter describes how to administer the Domain Name System (DNS). For more detailed information, see *DNS and Bind* by Cricket Liu and Paul Albitz, (O'Reilly, 1992) and "Name Server Operations Guide for BIND", University of California, Berkeley.

## Trailing Dots in Domain Names

When working with DNS−related files, follow these rules regarding the trailing dot in domain names:

- Use a trailing dot in domain names in hosts, hosts.rev, named.ca, and named.local data files. For example, sales.doc.com. is correct.
- Do not use a trailing dot in domain names in named.boot or resolv.conf files. For example, sales.doc.com is correct.

## Modifying DNS Data Files

Whenever you add or delete a host or make some other change in one of the DNS data files in the master DNS server or otherwise modify DNS data files, you must also:

- Change the serial number in the SOA resource record so the secondary servers modify their data accordingly (see *Changing the SOA Serial Number @ 2−1*).
- Inform in.named on the master server that it should reread the data files and update its internal database (see ).

### Changing the SOA Serial Number

Every DNS database file begins with a Start of Authority (SOA) resource record. Whenever you alter any data in a DNS database file, you must increment the SOA serial number by one integer.

For example, if the current SOA Serial Number in a data file is 101, and you make a change to the file's data, you must change 101 to 102. If you fail to change the SOA serial number, the domain's secondary servers will not update their copy of the database files with the new information and the primary and secondary servers will become out of synch.

A typical SOA record of a sample hosts file looks like this:

```
; sample  hosts  file
@ IN       SOA  nismaster.doc.com. root.nismaster.doc.com. (
   109 ; Serial
   10800 ; Refresh
                 1800 ; Retry
   3600000 ; Expire
   86400 ) ; Minimum
```

Thus, if you made a change to this hosts file, you would change 109 to 110. The next time you change the file, you would change 110 to 111.

## Forcing `in.named` to Reload DNS Data

When `in.named` successfully starts, the daemon writes its process ID to the file /etc/named.pid. To have `in.named` reread named.boot and reload the database, enter:

```
# kill −HUP ‘cat /etc/named.pid‘
```

This will eliminate all previously cache, and the caching process will start over again.

**Caution −** Do not attempt to run in.named from inetd. This will continuously restart the name server and defeat the purpose of having a cache.

## Adding and Deleting Machines

When you add or delete a machine, always make your changes in the data files stored on your primary DNS server. Do not make changes or edit the files on your secondary servers because those will be automatically updated from the primary server based on your changing the SOA serial number.

## Adding a Machine

To add a machine to a DNS domain, you set the new machine up as a DNS client and then add records for the new machine to the appropriate hosts and hosts.rev files.

For example, to add the host rigel to the doc.com domain:

1. **Create a /etc/resolv.conf file on rigel.**

2. **Add dns to the hosts line of rigel's /etc/nsswitch.conf file**

   (See *DNS and Internet Access @ 2−3*.)

3. **Add an address (A) record for rigel to the primary server's hosts file.**

   For example:
   ```
   rigel  IN  A  123.45.6.112
   ```

4. **Add any additional optional records for rigel to the primary server's hosts file.**

   Optional records could include:

   - Alias (CNAME)
   - Mail exchange (MX)
   - Well known services (WKS)
   - Host information (HINFO)

5. **Add a PTR record for rigel to the hosts.rev file.**
6. **Increment the SOA serial number in the primary server's hosts and hosts.rev files.**
7. **Reload the server's data.**

   Either reboot the server or enter:
   ```
   # kill −HUP ‘cat /etc/named.pid‘
   ```

   These steps are explained in more detail in *Solaris Naming Setup and Configuration Guide*.

## Removing a Machine

To remove a machine from a DNS domain:

1. **Remove dns from the hosts line of the machine's nsswitch.conf file.**
2. **Remove the machine's /etc/resolv.conf file.**
3. **Delete the records for that machine from the primary server's hosts and hosts.rev files.**
4. **If the machine has CNAME records pointing to it, those CNAME records must also be deleted from the hosts file.**
5. **Set up replacements for services supported by the removed machine.**

   If the machine is a primary server, mail host, or host for any other necessary process or service, you must take whatever steps are necessary to set up some other machine to perform those services.

## Adding Additional DNS Servers

You can add primary and secondary servers to your network. To add a DNS server:

1. **Set up the server up as a DNS client.**

See *Adding a Machine @ 2−1.*

2. **Set up the server's boot file.**
3. **Set up the server's named.ca file.**
4. **Set up the server's hosts file.**
5. **Set up the server's hosts.rev file.**
6. **Set up the server's named.local file.**
7. **Initialize the server.**
8. **Test the server.**

   These steps are explained in more detail in *Solaris Naming Setup and Configuration Guide*.

## Creating DNS Subdomains

As your network grows you may find it convenient to divide it into one or more DNS subdomains. (See *Introducing the DNS Namespace @ 1−2* for a discussion of DNS domain hierarchy and structure.)

When you divide your network into a parent domain and one or more subdomains, you reduce the load on individual DNS servers by distributing responsibility across multiple domains. In this way you can improve network performance. For example, suppose there are 900 machines on your network and all of them are in one domain. In this case, one set of DNS servers composed of a primary and additional secondary and caching−only servers have to support 900 machines. If you divide this network into a parent domain and two subdomain, each with 300 machines, then you have three sets of primary and secondary servers each responsible for only 300 machines.

By dividing your network into domains that match either your geographic or organizational structure (or both), the DNS domain names indicate where a given machine or email address fits into your structure. For example, rigel@alameda.doc.com implies that the machine rigel is located at your Alameda site, and the email address barnum@sales.doc.com implies that the user barnum is part of your Sales organization.

Dividing your network into multiple domains requires more set up work than keeping everything in one domain, and you have to maintain the delegation data that ties your domains together. On the other hand, when you have multiple domains, you can distribute domain maintenance tasks among different administrators or teams, one for each domain.

## Planning Your Subdomains

Here are some points to consider before dividing your network into a parent and one or more subdomains:

- *How many subdomains?* The more subdomains your create, the more initial set up work you have to do and the more ongoing coordination work for the administrators in the parent domain. The more subdomains, the more delegation work for the servers in the parent domain. On the other hand, fewer domains mean larger domains, and the larger a domain is the more server speed and memory is required to support it.

- *How to divide your network?* You can divide your network into multiple domains any way you want. The three most common methods are by organizational structure where you have separate subdomains for each department or division (sales, research, manufacturing, etc.); by geography where you have separate subdomains for each site; or by network structure where you have separate subdomains for each major network component. The most important rule to remember is that administration and use will be easier if your domain structure follows a consistent, logical, and self−evident pattern.
- *Consider the future.* The most confusing domain structures are those that grow over time with subdomains added haphazardly as new sites and departments are created. To the degree possible, try to take future growth into account when designing your domain hierarchy. Also take into account stability. It is best to base your subdomains on what is most stable. For example, if your geographic sites are relatively stable but your departments and divisions are frequently reorganized, it is probably better to base your subdomains on geography rather than organizational function. On the other hand, if your structure is relatively stable but you frequently add or change sites, it is probably better to base your subdomains on your organizational hierarchy.
- *Wide area network links.* When a network spans multiple sites connected via modems or leased lines, performance will be better and reliability greater if your domains do not span such Wide Area Network (WAN) links. In most cases, WAN links are slower than contiguous network connections and more prone to failure. When servers have to support machines that can only be reached over a WAN link, you increase the network traffic funneling through the slower link, and if there is a power failure or other problem at one site, it could affect the machines at the other sites. (The same performance and reliability considerations apply to DNS zones. As a general rule of thumb, it is best if zones do not span WAN links.)
- *NIS+ name service.* If your enterprise−level name service is NIS+, administration will be easier if your DNS and NIS+ domain and subdomain structures match.
- *Subdomain names.* To the degree possible, it is best to establish and follow a consistent policy for naming your subdomains. When domain names are consistent, it is much easier for users to remember and correctly specify them. Keep in mind that domain names are an important element in all of your DNS data files and that changing a subdomain name requires editing every file in which the old name appears. Thus, it is best to choose subdomain names that are stable and unlikely to need changing. You can use either full words, such as manufacturing, or abbreviations, such as manf, as subdomain names, but it will confuse users if some subdomains are named with abbreviations and others with full names. If you decide to use abbreviations, use enough letters to clearly identify the name because short cryptic names are hard to use and remember. Do not use reserved top−level Internet domain names as subdomain names. This means that names like org, net, com, gov, edu, and any of the two−letter country codes such as jp, uk, ca, and it should never be used as a subdomain name.

## Setting Up a Subdomain

In most cases, new subdomains are usually created from the start with a new network and machines, or split off from an existing domain. The process is essentially similar in both cases.

Once you have planned your new subdomain, follow these steps to set it up:

1. **Make sure all of the machines in the new subdomain are properly set up as DNS clients.**

   If you are carving a new subdomain out of an existing domain, most of the machines are probably already set up of DNS clients. If you are building a new subdomain from scratch (or adding new machines to an existing network) you must install properly configured resolv.conf and nsswitch.conf files on each machine as described in *Solaris Naming Setup and Configuration Guide*.

2. **Install properly configured boot and DNS data files on the subdomain's primary master server.**

   Install the following files on each server (see *Solaris Naming Setup and Configuration Guide* for details):

   - /etc/named.boot.
   - /var/named/named.ca.
   - /var/named/hosts.
   - /var/named/hosts.rev.
   - /var/named/named.local.

   Note that the server host files must have an Address (A) record, any necessary CNAME records for each machine in the subdomain and the server hosts.rev files must have a pointer (PTR) record for each machine in the subdomain. Optional HINFO and WKS records can also be added.

3. **If you are splitting an existing domain, remove the records for the machines in the new subdomain from the parent domain's master server hosts and hosts.rev files.**

   This requires deleting the A records for the machines that are now in the new subdomain from the hosts files of the old domain's servers, and also deleting the PTR records for those machines from the old domain's hosts.rev files. Any optional HINFO and WKS records for the moved machines should also be deleted.

4. **If you are splitting an existing domain, add the new subdomain name to CNAME records in the parent domain's master server hosts and file.**

   For example, suppose you are using the machine aldebaran as a fax server and it had the following CNAME record in the hosts file of the parent domain's servers:
   ```
   faxserver   IN   CNAME   aldebaran
   ```

   In addition to creating a new faxserver CNAME record for aldebaran in the hosts file of the new subdomain's master server, you would also have to change this CNAME record in the parent domain's hosts file to include aldebaran's subdomain as shown below:
   ```
   faxserver   IN   CNAME   aldebaran.manf.doc.com
   ```

5. **Add NS records for the new subdomain's servers to the parent domain's hosts file.**

   For example, suppose that your parent domain is doc.com and you are creating a new manf.doc.com subdomain with the machine rigel as manf's primary master server and aldebaran as the secondary master server. You would add the following records to the hosts file of doc.com's primary master server:
   ```
   manf.doc.com 99999 IN NS rigel.manf.doc.com
                99999 IN NS aldebaran.manf.doc.com
   ```

6. **Add A records for the new subdomain's servers to the parent domain's hosts file.**

   Continuing with the above example, you would add the following records to the hosts file of doc.com's primary master server:
   ```
   rigel.manf.doc.com       99999  IN  A  1.22.333.121
   ```

```
aldebaran.manf.doc.com   99999  IN  A  1.22.333.136
```

7. **Start up** `named` **on the subdomain's servers.**

   ```
   # /usr/sbin/in.named
   ```

   Instead of running `in.named` from the command line, you can reboot. See *Solaris Naming Setup and Configuration Guide* for details.

## DNS Error Messages and Problem Solving

See  *APPENDIX A*, *Problems and Solutions*, and  *APPENDIX B*, *Error Messages*," for DNS problem solving and error message information.