DNS for Rocket Scientists

This Open Source Guide is about DNS and (mostly) BIND 9.x on Linux (REDHAT Versions 6.x and 7.x) and the BSD's (FreeBSD, OpenBSD and NetBSD). It is meant for newbies, Rocket Scientist wannabees and anyone in between.

This Guide was born out of our first attempts a number of years ago at trying to install a much needed DNS service on an early Redhat Linux system. We completed the DNS 'rite of passage' and found it a pretty unedifying and pointless experience.

Health Warning: This is still a work-in-progress. If you have expertise in something - contribute some text. If you find errors don't grumble - tell us. Look at our to do list and if you want to contribute something please do so. And for all that hard work we promise only a warm sense of well-being and an acknowledgment of your work in the licence.

Section 1 Overview

What's new in Guide version 0.1.27

1. Boilerplate and Terminology

2. DNS - Overview

3. DNS Reverse Mapping

4. DNS Types

Section 2 - Get Something Running

5. BIND (Berkeley Internet Name Daemon)

One day real soon now ™

6. DNS Sample Configurations

Section 3 Mind Numbing Details

7. BIND named.conf Parameters

8. DNS Resource Records

Section 4 DNS Operations

Chapter 9 DNS HowTos

Chapter 10 Diagnostics and Tools

Chapter 11 Trouble and Error Messages

Work in progress

Chapter 12 BIND APIs

Work in progress

Section 5 DNS Security

Chapter 13 DNS Security

Section 6 DNS Bits and Bytes

Chapter 15 DNS Message Formats

Appendices: Resources

Maintenance Information

To do list - Stuff that still needs to be done.

Change log.

2. DNS Concepts

If you already understand what DNS is and does and how it fits into the greater scheme of things - skip this chapter.

2.1 A brief History of Name Servers

.. or why do we have DNS servers

Without a Name Service there would simply not be a viable Internet. To understand why we need to look at what DNS does and how and why it evolved.

1. A DNS translates (or maps) the name of a resource to its physical IP address
2. A DNS can also translate the physical IP address to the name of a resource by using reverse look-up or mapping.

Big deal.

Remember that the Internet (or any network for that matter) works by allocating every point (host, server, router, interface etc.) a physical IP address (which may be locally unique or globally unique).

Separation of Church and State.....

Without DNS every host (PC) which wanted to access a resource on the network (Internet), say a simple web page e.g. www.thing.com, would need to know its **physical IP address**. With 80 million'ish hosts and 20 million'ish web pages it is an impossible task - its also pretty impossible with just a handful of hosts and resources).

To solve this problem the concept of Name Servers was created in the mid 70's to enable certain attributes (properties) of a **named resource** to be maintained in a known location - the Name Server.

With a **Name Server** present in the network any host only needs to know the **physical address of a Name Server** and the **name** of the resource it wishes to access. Using this data it can find the address (or any other stored attribute or property) of the resource by interrogating (**querying**) the Name Server. Resources can be added, moved, changed or deleted at a single location - the Name Server. At a stroke network management was simplified and made more dynamic.

If it's broke....

We now have a new problem with our newly created Name Server concept. If our Name Server is not working our host cannot access any resource on the network. We have made the Name Server a critical resource. So we had better have more than one Name Server in case of failure.

To fix this problem the concept of Primary and Secondary Name Servers (many systems allow tertiary or more Name Servers) was born. If the Primary Name Server does not respond a host can use the Secondary (or tertiary etc.).

Man, we got more names than Webster....

As our network grows we start to build up a serious number of Names in our Name Server (database). This gives rise to three new problems.

1. Finding any entry in the database of names becomes increasingly slow as we power through many millions of names looking for the one we want. We need a way to index or organize the names.
2. If every host is accessing our Name Servers the load becomes very high. Maybe we need a way to spread the load across a number of servers.
3. With many Name (resource) records in our database the management problem becomes increasingly difficult as everyone tries to update all the records at the same time. Maybe we need a way to separate (or **delegate**) the administration of these Name (**resource**) records.

Which leads us nicely into the characteristics of the Internet's Domain Name System (DNS).

2.2 DNS Concepts & Implementation

The Internet's Domain Name Service (DNS) is just a specific implementation of the Name Server concept optimized for the prevailing conditions on the Internet.

2.2.1 DNS Overview

From our brief history of Name Servers we saw how three needs emerged:

1. The need for a hierarchy of names
2. The need to spread the operational loads on our name servers
3. The need to delegate the administration of our Name servers

The Internet Domain Name System elegantly solves all these problems at the single stoke of a pen (well actually the whole of RFC 1034 to be precise).

2.2.2 Domains and Delegation

The Domain Name System uses a tree (or hierarchical) name structure. At the top of the tree is the root followed by the Top Level Domains (TLDs) then the domain-name and any number of lower levels each separated with a dot.

NOTE: The root of the tree is represented most of the time as a silent dot ('.') but there are times as we shall see later when it VERY important.

Top Level Domains (TLDs) are split into two types:

1. Generic Top Level Domains (gTLD) .com, .edu, .net, .org, .mil etc.
2. Country Code Top Level Domain (ccTLD) e.g. .us, .ca, .tv , .uk etc.

Country Code TLDs (ccTLDs) use a standard two letter sequence defined by ISO 3166.

Figure 1-1 shows this diagrammatically.

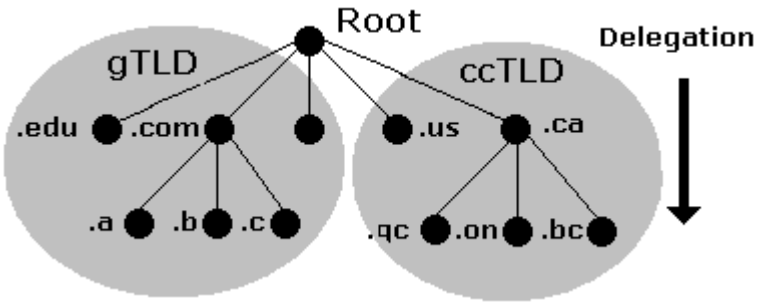


Figure 1-1 Domain Structure and Delegation

What is commonly called a 'Domain Name' is actually a combination of a domain-name and a TLD and is written from LEFT to RIGHT with the lowest level in the hierarchy on the left and the highest level on the right.

```
domain-name.tld e.g. example.com
```

In the case of the gTLDs e.g. .com, .net etc. the user part of the delegated name - the name the user registered - is called a Second Level Domain (SLD), it is the second level in the hierarchy. The user part is frequently simply referred to as the SLD. So the the Domain Name in the example above can be re-defined to consist of:

```
sld.tld e.g. example.com
```

The term Second Level Domain (SLD) is much less useful with ccTLDs where the user registered part is frequently the Third Level Domain e.g.:

```
example.co.uk
example.com.br
```

The term Second Level Domain (SLD) provides technical precision but can be confusing - unless the precision is required we will continue to use the generic term Domain Name or simply Domain to the whole name e.g. a Domain Name is example.com or example.co.uk.

Authority and Delegation

The concepts of **Delegation** and **Authority** lie at the core of the domain name system hierarchy. The **Authority** for the root domain lies with Internet Corporation for Assigned Numbers and Names (ICANN). Since 1998 ICANN, a non-profit organisation, has assumed this responsibility from the US government.

The gTLDs are **authoritatively** administered by ICANN and delegated to a series of accredited registrars. The ccTLDs are delegated to the individual countries for administration purposes. Figure 1.0 above shows how any authority may in turn delegate to lower levels in the hierarchy, in other words it may delegate anything for which it is **authoritative**. Each layer in the hierarchy may **delegate** the **authoritative** control to the next lower level.

In the case of ccTLDs countries like Canada (ccTLD .ca) and the US (ccTLD .us) and others with federal governments have decided that they will administer at the national level and delegate to each province (Canada) or state (US) a two character province/state code. e.g. .qc = Quebec, .ny = New York, md = Maryland etc.. Thus mycompany.md.us is the Domain Name of 'mycompany' which was delegated from the state of MaryLand in the US.

Countries with more centralized governments, like the UK and others, have opted for functional segmentation in their delegation models e.g. .co = company, .ac = academic etc.). Thus mycompany.co.uk is the 'Domain Name' of 'mycompany' registered as a company from the UK registration authority.

Delegation within any domain may be almost limitless and is **decided by the delegated authority** e.g. the US and Canada both delegate city within province/state domains e.g. the address (or URL) tennisshoes.nb.us is the town of Tennis Shoes in the State of Nebraska in the United States.

By reading a domain name from RIGHT to LEFT you can track its delegation. This unit of delegation is usually referred to as a 'zone' in standards documentation.

So What is www.example.com

From our reading above we can see that www.example.com is built up from 'www' and 'example.com'. The 'Domain-Name' example.com part was delegated from a registrar which in turn was delegated from ICANN.

The 'www' part was chosen by the owner of the domain since they are now the delegated authority for the 'example.com' name. They own EVERYTHING to the LEFT of the delegated 'Domain Name'.

The leftmost part, the 'www' in this case, is called a host name. By convention (but only convention) web sites have the 'host' name of www (for world wide web) but you can have a web site whose name is fred.example.com - no-one may think of typing this into their browser but that does not stop you doing it!

Every computer that is connected to the internet or an internal network has a host name, here are some more examples:

```
www.example.com - the company web service
ftp.example.com - the company file transfer protocol server
pc17.example.com - a normal PC
accounting.example.com - the main accounting system
```

A host name must be unique within the 'Domain Name' but can be anything the owner of 'example.com' wants.

Finally lets look at this name:

```
www.us.example.com
```

From our previous reading we figure its 'Domain Name' is example.com the 'www' probably indicates a web site which leaves the 'us' part.

The 'us' part was allocated by the owner of 'mydomain.com (they are authoritative) and is called a sub-domain. In this case the **delegated authority** for example.com has decided that their company organization is best served by a country based sub-domain structure. They could have **delegated** the responsibility internally to the US subsidiary for administration of this sub-domain, which may in turn have created a plant based structure e.g. www.cleveland.us.example.com could indicate the web site of the Cleveland plant in the US organisation of 'example.com'.

To summarise the OWNER can delegate, IN ANY WAY THEY WANT, ANYTHING to the LEFT of the 'Domain Name' they own (were delegated). The owner is also RESPONSIBLE for administering this delegation.

Note: Names such as www.example.com and www.us.example.com are commonly - but erroneously - referred to as Fully Qualified Domain Names (FQDN). Technically an FQDN unambiguously defines a name from *any starting point* to the root and as such must contain the normally silent dot at the end e.g. "www.example.com." is an FQDN "www.example.com" is not.

2.2.3 DNS Organization and Structure

The Internet's DNS exactly maps the 'Domain Name' delegation structure described above. There is a DNS server running at each level in the delegated hierarchy and the responsibility for running the DNS lies with the AUTHORITATIVE control at that level.

Figure 1-2 shows this diagrammatically.

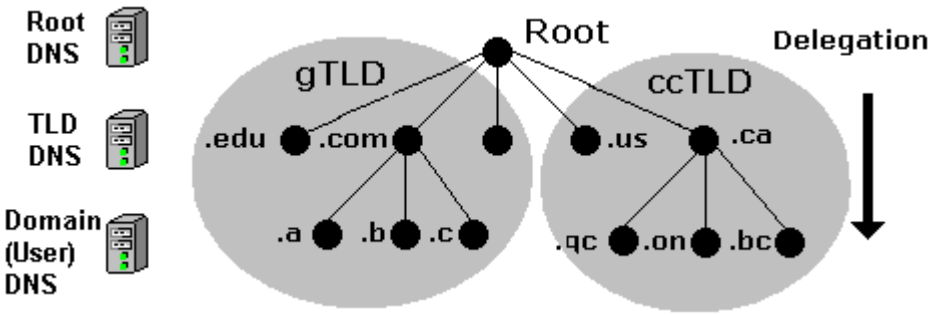


Figure 1-2 DNS mapped to Domain Delegation

The Root Servers (Root DNS) are the responsibility of ICANN but operated by a consortium under a delegation agreement. ICANN created the Root Servers Systems Advisory Committee (RSSAC) to provide advice and guidance as to the operation and development of this critical resource. The IETF was requested by the RSSAC to develop the engineering standards for operation of the Root-Servers. This request resulted in the publication of RFC 2870.

There are currently (mid 2003) 13 root-servers world-wide. The Root-Servers are known to every public DNS server in the world.

The TLD servers (ccTLD and gTLD) are operated by a variety of agencies and registrars under a fairly complex set of agreements by **Registry Operators**.

The Authority and therefore the responsibility for the User (or 'Domain Name') DNS servers lies with the owner of the domain. In many cases this responsibility is delegated by the owner of the Domain to an ISP, Web Hosting company or increasingly a registrar. Many companies, however, elect to run their own DNS

servers and even delegate the Authority and responsibility for sub-domain DNS servers to separate parts of the organisation.

When any DNS cannot answer (resolve) a request for a domain name from a host e.g. example.com the query is passed to a **root-rerver** which will direct the query to the appropriate TLD DNS server which will in turn direct it to the appropriate Domain (User) DNS server.

2.2.4 DNS System Components

A Domain Name System (DNS) as defined by RFC 1034 includes three parts:

1. Data which describes the domain(s)
2. One or more Name Server programs.
3. A resolver program or library.

A single DNS server may support many domains. The data for each domain describes global properties of the domain and its hosts (or services). This data is defined in the form of textual Resource Records organized in Zone Files. The format of Zone files is defined in RFC 1035 and is supported by most DNS software.

The Name Server program typically does three things:

1. It will read a configuration file which defines the zones for which it is responsible.
2. Depending on the Name Servers functionality the configuration file may describe various behaviours e.g. to cache or not. Some DNS servers are very specialized and do not provide this level of control.
3. Respond to questions (queries) from local or remote hosts.

The resolver program or library is located on each host and provides a means of translating a users request for, say, www.thing.com into one or more queries to DNS servers using UDP (or TCP) protocols.

Note: The resolver on all Windows systems and the majority of *nix systems is actually a **stub** resolver - a minimal resolver that can only work with a DNS that supports recursive queries. The caching resolver on MS Windows 2K and XP is a **stub** resolver with a cache to speed up responses and reduce network usage.

While BIND is the best known of the DNS servers and much of this guide documents BIND features, it is by no means the only solution or for that matter the only Open Source solution. Appendix C: lists many alternate solutions. The zone file formats which constitute the majority of the work (depending on how many sites you operate) is standard (defined by RFC 1035) and is typically supported by all DNS suppliers. Where a feature is unique to BIND we indicate it clearly in the text so you can keep your options **open**!

2.2.5 Zones and Zone Files

Zone files contain Resource Records that describe a domain or sub-domain. The format of zone files is defined by RFC 1035 and is an IETF standard. Almost any sensible DNS software should be able to read zone files. A zone file will consist of the following types of data:

1. Data that describes the top of the zone (a SOA Record).
2. Authoritative data for all nodes or hosts within the zone (typically A Records).
3. Data that describes global information for the zone (typically MX Records and NS Records).
4. In the case of sub-domain delegation the name servers responsible for this sub-domain (a NS Record).
5. In the case of sub-domain delegation a 'glue' record that allows this name server to reach the sub-domain (typically one or more A Records) for the sub-domain name servers.

The individual Resource Records are described and numerous sample configuration files are illustrated and described.

2.2.6 DNS Queries

The major task carried out by a DNS server is to respond to queries (questions) from a local or remote resolver or other DNS acting on behalf of a resolver. A query would be something like 'what is the IP address of fred.example.com'.

A DNS server may receive such a query for any domain. DNS servers may be configured to be authoritative for some (if any) domains, slaves, caching, forwarding or many other combinations for others.

Most of the queries that a DNS server will receive will be for domains for which it has no knowledge i.e for which it has no local zone files. The DNS sofware typically allows the name server to respond in different ways to queries about which it has no knowledge.

There are three types of queries defined for DNS:

1. A recursive query - the complete answer to the question is always returned. DNS servers are not required to support recursive queries.
2. An Iterative (or non-recursive) query - where the complete answer MAY be returned. All DNS servers must support Iterative queries.
3. An Inverse query - where the user wants to know the domain name given a resource record.

Note: The process called Reverse Mapping (returns a host name given an IP address) does not use Inverse queries but instead uses Recursive and Iterative (non-recursive) queries using the special domain name IN-ADDR.ARPA.

Historically reverse IP mapping was not mandatory. Many systems however now use reverse mapping for security and simple authentication schemes so proper implementation and maintenance is now practically essential.

2.2.6.1 Recursive Queries

A recursive query is one where the DNS server will fully answer the query (or give an error). DNS servers are not required to support recursive queries and both the resolver (or another DNS acting recursively on behalf of another resolver) negotiate use of recursive service using bits in the query headers.

There are three possible responses to a recursive query:

1. The answer to the query accompanied by any CNAME records (aliases) that may be useful. The response will indicate whether the data is authoritative or cached.
2. An error indicating the domain or host does not exist (NXDOMAIN). This response may also contain CNAME records that pointed to the non-existing host.
3. An temporary error indication - e.g. can't access other DNS's due to network error etc..

In a recursive query a DNS server will, on behalf of the client (resolver), chase the trail of DNS across the universe to get the real answer to the question. The journey of a simple query such as 'what is the IP address of fred.example.com' to a DNS server which supports recursive queries but is not authoritative for example.com could look something like this:

1. Resolver on a host sends query 'what is the IP address of fred.example.com' to locally configured DNS server.
2. DNS server looks up fred.example.com in local tables (its **cache**) - not found
3. DNS sends query to a root-server for the IP of fred.example.com
4. The root-server replies with a referral to the TLD servers for .com
5. The DNS server sends query 'what is the IP address fred.example.com' to .com TLD server.
6. The TLD server replies with a referral to the name servers for example.com
7. The DNS server sends query 'what is the IP address fred.example.com' to name server for example.com.
8. Zone file defines a CNAME record which shows fred is aliased to joe. DNS returns both the CNAME and the A record for joe.
9. send response joe=x.x.x.x (with CNAME record fred=joe) to original client resolver. Transaction complete.

2.2.6.2 Iterative (non-recursive) Queries

A Iterative (or non-recursive) query is one where the DNS server may provide a partial answer to the query (or give an error). DNS servers must support non-recursive queries.

There are four possible responses to a non-recursive query:

1. The answer to the query accompanied by any CNAME records (aliases) that may be useful. The response will indicate whether the data is authoritative or cached.
2. An error indicating the domain or host does not exist (NXDOMAIN). This response may also contain CNAME records that pointed to the non-existing host.
3. An temporary error indication - e.g. can't access other DNS's due to network error etc..
4. A **referral**; the name and IP addess(es) or one or more name server(s) that are closer to the requested domain name. This may, or may not be, the authoritative name server for the target domain.

The journey of a simple query such as 'what is the IP address of fred.example.com' to a DNS server which supports Iterative (non-recursive) queries but is not authoritative for example.com could look something like this:

1. Resolver on a host sends query 'what is the IP address fred.example.com' to locally configured DNS server.
2. DNS server looks up fred.example.com in local tables (its **cache**) - not found
3. The DNS replies with a referral containing the root-servers
4. Resolver sends query to a root-server for the IP of fred.example.com
5. The root-server replies with a referral to the TLD servers for .com
6. The Resolver sends query 'what is the IP address fred.example.com' to .com TLD server.
7. The TLD server replies with a referral to the name servers for example.com
8. The Resolver sends query 'what is the IP address fred.example.com' to name server for example.com.
9. Zone file defines a CNAME record which shows fred is aliased to joe. DNS returns both the CNAME and the A record for joe.
10. Transaction complete.

Note: The above sequence is highly artificial since the resolver on Windows and most *nix systems is a **stub** resolver - which is defined in the standards to be a minimal resolver which cannot follow **referrals**. If you reconfigure your local PC or Workstation to point to a DNS server that only supports Iterative queries - it will not work. Period.

2.2.6.3 Inverse Queries

An Inverse query maps a resource record to a domain. An example Inverse query would be 'what is the domain name for this MX record'. Inverse query support is optional and it is permitted for the DNS server to return a response **Not Implemented**.

Inverse queries are NOT used to find a host name given an IP address. This process is called Reverse Mapping (Look-up) uses recursive and Iterative (non-recursive) queries with the special domain name IN-ADDR.ARPA.

2.2.7 Zone Updates

The initial design of DNS allowed for changes to be propagated using Zone Transfer (AXFR) but the world of the Internet was simpler and more sedate in those days (1987). The desire to speed up the process of zone update propagation while minimising resources used has resulted in a mumber of changes to this aspect of DNS design and implementation from simple - but effective - tinkering such as **Incremental Zone Transfer (IXFR)** and **Notify** messages to the concept of **Dynamic Updates** which is still not widely deployed.

Warning While zone transfers are generally essential for the operation of DNS systems they are also a source of threat. A slave DNS can become **poisoned** if it accepts zone updates from a malicious source. Care should be taken during configuration to ensure that, as a minimum, the 'slave' will only accept transfers from known sources. The example configurations provide these minimum precautions. Security Overview outlines some of the potential threats involved.

2.2.7.1 Full Zone Update (AXFR)

The original DNS specifications (RFC 1034 & RFC 1035) envisaged that slave (or secondary) DNS servers would 'poll' the 'master'. The time between such 'polling' is determined by the REFRESH value on the domain's SOA Resource Record

The polling process is accomplished by the 'slave' sending a query to the 'master' and requesting the latest SOA record. If the SERIAL number of the record is different from the current one maintained by the 'slave' a zone transfer (AXFR) is requested. This why it is vital to very disciplined about updating the SOA serial number every time anything changes in ANY of the zone records.

Zone transfers are always carried out using TCP on port 53 not UDP (normal DNS query operations use UDP on port 53).

2.2.7.2 Incremental Zone Update (IXFR)

Transferring very large zone files can take a long time and waste bandwidth and other resources. This is especially wasteful if only a single record has been changed! RFC 1995 introduced Incremental Zone Transfers (IXFR) which as the name suggests allows the 'slave' and 'master' to transfer only those records that have changed.

The process works as for AXFR. The 'slave' sends a query for the domain's SOA Resource Record every REFRESH interval. If the SERIAL value of the SOA record has changed the 'slave' requests a Zone Transfer and indicates whether or not it is capable of accepting an Incremental Transfer (IXFR). If both 'master' and 'slave' support the feature an Incremental Transfer (IXFR) takes place otherwise a Full Zone Transfer (AXFR) takes place. Incremental Zone transfers use TCP on port 53 (normal DNS queries operations use UDP on port 53).

The default mode for BIND when acting as a 'slave' is to use IXFR unless it is configured not to using the **request-ixfr** parameter in the **server** or **options** section of the named.conf file.

The default mode for BIND when acting as a 'master' is to use IXFR only when the zone is dynamic. The use of IXFR is controlled using the **provide-ixfr** parameter in the **server** or **options** section of the named.conf file.

2.2.7.3 Notify (NOTIFY)

RFC 1912 recommends a REFRESH interval of up to 12 hours on the REFRESH interval of an SOA Resource Record. This means that changes to the 'master' DNS may not be visible at the 'slave' DNS for up to 12 hours. In a dynamic environment this may be unacceptable.

RFC 1996 introduced a scheme whereby the **master** will send a NOTIFY message to the **slave** DNS systems that a change MAY have occurred in the domain records. The 'slave' on receipt of the NOTIFY will request the latest SOA Resource Record and if the SERIAL value is different will attempt a Zone Transfer using either a full Zone Transfer (AXFR) or an Incremental Transfer (IXFR).

NOTIFY behaviour in BIND is controlled by **notify**, **also-notify** and **notify-source** parameters in the **zone** or **options** statements of the named.conf file.

2.2.7.4 Dynamic Update

The classic method of updating Zone Resource Records is to manually edit the zone file and then stop and start the name server to propagate the changes. When the volume of changes reaches a certain level this can become operationally unacceptable - especially considering that in organisations which handle large numbers of Zone Files, such as service providers, BIND itself can take a long time to restart at it plows through very large numbers of zone statements.

The 'holy grail' of DNS is to provide a method of dynamically changing the DNS records while DNS continues to service requests.

There are two architectural approaches to solving this problem:

1. Allow 'run-time' updating of the Zone Records from an external source/application.
2. directly feed BIND (say via one of its two APIs) from a database which can be dynamically updated.

RFC 2136 takes the first approach and defines a process where zone records can be updated from an external source. The key limitation in this specification is that a new domain cannot be added dynamically. All other records within an existing zone can be added, changed or deleted. In fact this limitation is also true for both of BIND's APIs as well.

As part of this specification the term **Primary Master** is coined to describe the Name Server defined in the SOA Resource Record for the zone. The significance of this term is that when dynamically updating records it is essential to update only one server even though there may be multiple **master** servers for the zone. In order to solve this problem a 'boss' server must be selected, this 'boss' server termed the **Primary Master** has no special characteristics other than it is defined as the Name Server in the SOA record and may appear in an **allow-update** clause to control the update process.

While normally associated with Secure DNS features (TSIG - RFC 2845 & TKEY - RFC 2930) Dynamic DNS (**DDNS**) does not REQUIRE TSIG/TKEY. However there is a good reason to associate the two specifications when you consider that by enabling Dynamic DNS you are opening up the possibility of **master** zone file corruption or poisoning. Simple IP address protection (acl) can be configured into BIND but this provides - at best - limited protection. For that reason serious users of Dynamic DNS will always use TSIG/TKEY procedures to authenticate incoming requests.

Dynamic Updating is defaulted to **deny from all hosts**. Control of Dynamic Update is provided by the BIND **allow-update** (usable with and without TSIG/TKEY) and **update-policy** (only usable with TSIG/TKEY) clauses in the **zone** or **options** statements of the named.conf file.

There are a number of Open Source tools which will initiate Dynamic DNS updates these include **dnsupdate** (not the same as DNSUpdate) and **nsupdate** which is distributed with **bind-utils**.

2.2.7.5 Alternative Dynamic DNS Approaches

As noted above the major limitation in the standard Dynamic DNS (RFC 2136) approach is that new domains cannot be created dynamically.

BIND-DLZ takes a much more radical approach and using a serious patch to BIND allows replacement of all zone files with a single zone file which defines a database entry. The database support, which includes most of the major databases (MySQL, PostgreSQL, BDB and LDAP among others) allows the addition of new domains as well as changes to pre-existing domains without the need to stop and start BIND. As with all things in life there is a trade-off and performance can drop precipitously.

Current work being carried (early 2004) out with a High performance Berkeley DB (BDB) is showing excellent results approaching raw BIND performance.

PowerDNS an authoritative only name server takes a similar approach with its own (non-BIND) code base by referring all queries to the database back-end and thereby allow new domains to be added dynamically.

2.3 Security Overview

DNS Security is a huge and complex topic. It is made worse by the fact that almost all the documentation dives right in and you fail to see the forest for all the d@!mned trees.

The critical point is to first understand what you want to secure - or rather what threat level you want to secure against. This will be very different if you run a root server vs running a modest in-house DNS serving a couple of low volume web sites.

The term DNSSEC is thrown around as a blanket term in a lot of documentation. This is not correct. There are at least three types of DNS security, two of which are - relatively - painless and DNSSEC which is - relatively - painful.

Security is always an injudicious blend of real threat and paranoia - but remember just because you are naturally paranoid does not mean that **they** are not after you!

2.3.1 Security Threats

In order to be able to assess both the potential threats and the possible counter-measures it is first and foremost necessary to understand the normal data flows in a DNS system. Diagram 1-3 below shows this flow.

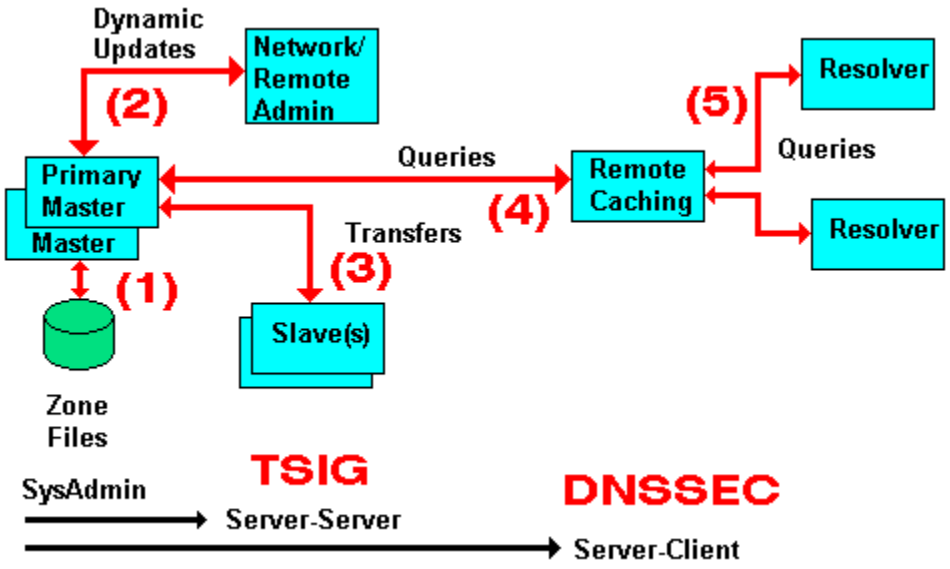


Diagram 1-3 DNS Data Flow

Every data flow (each RED line above) is a potential source of threat! Using the numbers from the above diagram here is what can happen at each flow - beware you may not sleep tonight:

Number	Area	Threat
(1)	Zone Files	File Corruption (malicious or accidental). Local threat.
(2)	Dynamic Updates	Unauthorized Updates, IP address spoofing (impersonating update source). Server to Server (TSIG Transaction) threat.
(3)	Zone Transfers	IP address spoofing (impersonating update source). Server to Server (TSIG Transaction) threat.
(4)	Remote Queries	Cache Poisoning by IP spoofing, data interception, or a subverted Master or Slave. Server to Client (DNSSEC) threat.
(5)	Resolver Queries	Data interception, Poisoned Cache, subverted Master or Slave, local IP spoofing. Remote Client-client (DNSSEC) threat.

The first phase of getting a handle on the problem is to figure (audit) what threats are applicable and how seriously do YOU rate them or do they even apply. As an example; if you don't do Dynamic Updates (BIND's default mode) - there is no Dynamic Update threat! Finally in this section a warning: **the further you go from the Master the more complicated the solution and implementation**. Unless there is a very good reason for not doing so, we would always recommend that you start from the Master and work out.

2.3.2 Security Types

We classify each threat type below. This classification simply allows us select appropriate remedies and strategies for avoiding or securing our system. The numbering used below relates to diagram 1-3.

(1) The primary source of Zone data is normally the Zone Files (and don't forget the named.conf file which contains lots of interesting data as well). This data should be secure and securely backed up. This threat is classified as **Local** and is typically handled by good system administration.

(2) If you run slave servers you will do zone transfers. **Note:** You do NOT have to run with slave servers, you can run with multiple masters and eliminate the transfer threat entirely. This is classified as a **Server-Server (Transaction)** threat.

(3) The BIND default is to **deny** Dynamic Zone Updates. If you have enabled this service or require to it poses a serious threat to the integrity of your Zone files and should be protected. This is classified as a **Server-Server (Transaction)** threat.

(4) The possibility of Remote Cache Poisoning due to IP spoofing, data interception and other hacks is a judgement call if you are running a simple web site. If the site is high profile, open to competitive threat or is a high revenue earner you have probably implemented solutions already. This is classified as a **Server-Client** threat.

(5) We understand that certain groups are already looking at the implications for secure Resolvers but as of early 2004 this was not standardised. This is classified as a **Server-Client** threat.

2.3.3 Security - Local

Normal system administration practices such as ensuring that files (configuration and zone files) are securely backed-up, proper read and write permissions applied and sensible physical access control to servers may be sufficient.

Implementing a Stealth (or Split) DNS server provides a more serious solution depending on available resources.

Finally you can run BIND (named) in a **chroot jail**.

2.3.4 Server-Server (TSIG Transactions)

Zone transfers. If you have slave servers you will do zone transfers. BIND provides **Access Control Lists (ACLs)** which allow simple IP address protection. While IP based **ACLs** are relatively easy to subvert they are a **lot** better than nothing and require very little work. You can run with multiple masters (no slaves) and eliminate

the threat entirely. You will have to manually synchronise zone file updates but this may be a simpler solution if changes are not frequent.

Dynamic Updates. If you must run with this service it should be secured. BIND provides **Access Control Lists (ACLs)** which allow simple IP address protection but this is probably not adequate unless you can secure the IP addresses i.e. both systems are behind a firewall/DMZ/NAT or the updating host is using a private IP address.

TSIG/TKEY If all other solutions fail DNS specifications (RFC 2845 - TSIG and RFC 2930 - TKEY) provide authentication protocol enhancements to secure these Server-Server transactions.

TSIG and **TKEY** implementations are messy but not too complicated - simply because of the scope of the problem. With **Server-Server** transactions there is a finite and normally small number of hosts involved. The protocols depend on a **shared secret** between the master and the slave(s) or updater(s). It is further assumed that you can get the **shared secret** securely to the peer server by some means not covered in the protocol itself. This process, known as **key exchange**, may not be trivial (typically long random strings of base64 characters are involved) but you can use the telephone(!), mail, fax or PGP email amongst other methods.

The **shared-secret** is open to **brute-force** attacks so frequent (monthly or more) changing of **shared secrets** will become a fact of life. What works once may not work monthly or weekly. **TKEY** allows automation of **key-exchange** using a Diffie-Hellman algorithm but seems to start with a **shared secret**!

2.3.5 Server-Client (DNSSEC)

The classic Remote Poisoned cache problem is not trivial to solve simply because there may be an infinitely large number of Remote Caches involved. It is not reasonable to assume that you can use a **shared secret**.

Instead **DNSSEC** relies on **public/private key authentication**. The DNSSEC specifications (RFC 2535 augmented with others) attempt to answer three questions:

1. Authentication - the DNS responding really is the DNS that the request was sent to.
2. Integrity - the response is complete and nothing is missing.
3. Integrity - the DNS records have not been compromised.

3. DNS Reverse Mapping

- 3.3 Reverse Map Delegation

3.1 Reverse Mapping Overview

A normal DNS query would be of the form 'what is the IP of host=www in domain=mydomain.com'. There are times however when we want to be able to find out the name of the host whose IP address = x.x.x.x. Sometimes this is required for diagnostic purposes more frequently these days it is used for security purposes to trace a hacker or spammer, indeed many modern mailing systems use reverse mapping to provide simple authentication using dual look-up, IP to name and name to IP.

In order to perform Reverse Mapping and to support normal recursive and Iterative (non-recursive) queries the DNS designers defined a special (reserved) **Domain Name** called IN-ADDR.ARPA. This domain allows for all supported Internet IPv4 addresses (and now IPv6).

3.2 IN-ADDR.ARPA Reverse Mapping Domain

Reverse Mapping looks horribly complicated. It is not. As with all things when we understand what is being done and why - all becomes as clear as mud!

We defined the normal domain name structure as a tree starting from the root. We write a normal domain name LEFT to RIGHT but the hierarchical structure is RIGHT to LEFT.

```
domain name = www.mydomain.com
highest node in tree is = .com
next (lower) = .mydomain
next (lower) = www
```

An IPv4 address is written as:

```
192.168.23.17
```

This IPv4 address defines a host = 17 in a Class C address range (192.168.23.x). In this case the most important part (the highest node) is on the LEFT (192) not the RIGHT. This is a tad awkward and would make it impossible to construct a sensible tree structure that could be searched in a single lifetime.

The solution is to reverse the order of the address and place the result under the special domain IN-ADDR.ARPA (you will see this also written as in-addr.arpa which is OK since domains are case insensitive but the case should be preserved so we will use IN-ADDR.ARPA).

Finally the last part of the IPv4 Address (17) is the host address and hosts, from our previous reading, are typically defined inside a zone file so we will ignore it and only use the Class C address base. The result of our manipulations are:

```
IP address =192.168.23.17
Class C base = 192.168.23 ; omits the host address = 17
Reversed Class C base = 23.168.192
Added to IN-ADDR.ARPA domain = 23.168.192.IN-ADDR.ARPA
```

This is show in figure 3.0 below.

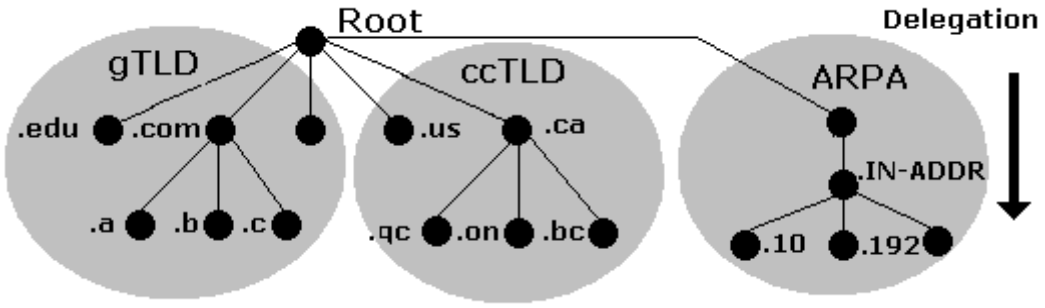


Figure 3.0 IN-ADDR.ARPA Reverse Mapping</B< p>

Finally we construct a zone file to describe all the hosts (nodes) in the Reverse Mapped zone using PTR Records. The resulting file will look something like this:

```
$ORIGIN 23.168.192.IN-ADDR.ARPA.
@             IN      SOA   ns1.foo.com. root.foo.com. (
                              2003080800 ; serial number
                              3h         ; refresh
                              15m        ; update retry
                              3w         ; expiry
                              3h         ; minimum
                              )
              IN      NS      ns1.foo.com.
              IN      NS      ns2.foo.com.
1             IN      PTR     www.foo.com. ; qualified name
2             IN      PTR     joe.foo.com.
.....
17            IN      PTR     bill.foo.com.
.....
74            IN      PTR     fred.foo.com.
....
```

We must use qualified names ending with a dot (in fact they are Fully Qualified Domain Names FQDN) in this file because if we did not our $ORIGIN directive would lead to some strange results.

3.3 Reverse Map Delegation

Classless Reverse Map Delegation is defined by RFC 2317 which has Best Current Practice status and should be regarded as a definitive reference. Classless routing allows allocation of sub-nets on non-octet boundaries i.e. less that 256 addresses from a Class C address may be allocated and routed. The technique defined in the RFC is attributed to Glen A. Herrmannsfeldt.

Normal domain name mapping as we have seen maps the domain name to an IP address. This process is independent of the ISP or other authority that allocated the IP name space. If the addresses were to change then the owner of the domain that maps these addresses would be able to make the necessary changes directly with either the relevant registrar i.e. change the IP address of DNS's for the domain or change the zone file(s) that describe the domain.

The rule is that entities can be delegated only once in the domain name tree this includes IN-ADDR.ARPA. When a Class C subnet is assigned by an ISP or other authority e.g. 192.168.23.64/27 (a 32 IP address subnet) the responsibility for reverse mapping for the whole Class C address has already been assigned to the ISP or Authority. If you want to change the host names in the assigned subnet they must be notified to the authority for that Class C address. Generally this is unacceptable since such requests may encounter indifference, cost or questions. It is most desirable that responsibility for reverse mapping be delegated when the IP address subnet is assigned.

The technique defined in RFC 2317 provides for such delegation to take place using CNAME Resource Records (rather than the more normal PTR Resource Records) in an expanded IN-ADDR.ARPA name space.

The following fragment shows our 192.168.23.64/27 subnet as a fragment of the reverse mapping zone file located at the ISP or other Authority that assigned the subnet:

```
$ORIGIN 23.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.isp.com. root.isp.com. (
                              2003080800 ; serial number
                              3h         ; refresh
                              15m        ; update retry
                              3w         ; expiry
                              3h      ; minimum
                              )
              IN  NS      ns1.isp.com.
              IN  NS      ns2.isp.com.
; definition of other IP address 0 - 63
....

; definition of our target 192.168.23.64/27 subnet
; name servers for subnet reverse map
64/27         IN  NS  ns1.mydomain.com.
64/27         IN  NS  ns2.mydomain.com.
```

```
; IPs addresses in the subnet - all need to be defined
; except 64 and 95 since they are the subnets
; broadcast and multicast addresses not hosts/nodes
65            IN  CNAME   65.64/27.23.168.192.IN_ADDR.ARPA. ;qualified
66            IN  CNAME   66.64/27 ;unqualified name
67            IN  CNAME   67.64/27
....
93            IN  CNAME   93.64/27
94            IN  CNAME   94.64/27
; end of 192.168.23.64/27 subnet
.....
; other subnet definitions
```

The 64/27 construct is an artificial (but legitimate) way of constructing the additional space to allow delegation. This is not technically a domain name and therefore can use '/' (which is not allowed in a domain name) but could be replaced with say '-' which is allowed e.g. 64-27.

The zone file at the DNS serving the Reverse Map (ns1.mydomain.com in the above example) looks like this:

```
$ORIGIN 64/27.23.168.192.IN-ADDR.ARPA.
@             IN  SOA   ns1.mydomain.com. root.mydomain.com. (
                              2003080800 ; serial number
                              3h         ; refresh
                              15m        ; update retry
                              3w         ; expiry
                              3h      ; minimum
                              )
              IN  NS      ns1.mydomain.com.
              IN  NS      ns2.mydomain.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 95 since they are the subnets
; broadcast and multicast addresses not hosts/nodes
65            IN  PTR   fred.mydomain.com. ;qualified
66            IN  PTR   joe.mydomain.com.
67            IN  PTR   bill.mydomain.com.
....
93            IN  PTR   web.mydomain.com.
94            IN  PTR   ftp.mydomain.com.
; end of 192.168.23.64/27 subnet
```

Now you have to change your reverse map zone names in the name.conf file to reflect the above change. The following examples shows the reverse map declaration before and after the change to reflect the configuration above:

```
//  before change the reverse map zone declaration would look
// something like this
zone "23.168.192.in-addr.arpa" in{
        type master;
        file "192.168.23.rev";
};
```

The above - normal - reverse map declaration resolves reverse lookups for 192.168.23.x locally and without the need for access to any other zone or DNS.

Change to reflect the delegated zone name.

```
//  after change the reverse map zone declaration would look
// something like this
zone "64/27.23.168.192.in-addr.arpa" in{
        type master;
        file "192.169.23.rev";
};
```

The above configuration will only resolve by querying the master zone for 23.168.192.IN-ADDR.ARPA and following down the delegation back to itself. If changes are not made at the ISP or issuing Authority or have not yet propagated then this configuration will generate 'nslookup' and 'dig' errors.

4. DNS Configuration Types

Most DNS servers are schizophrenic - they may be masters (authoritative) for some zones, slaves for others and provide caching or forwarding for all others. Many observers object to the concept of DNS **types** partly because of the schizophrenic behaviour of most DNS servers and partly to avoid confusion with the name.conf zone parameter 'type' which only allows master, slave, stub, forward, hint). Nevertheless, the following terms are commonly used to describe the primary function or requirement of DNS servers.

Contents

4.1 Master (Primary) Name Servers

A Master DNS contains one or more zone files for which this DNS is **Authoritative** ('type master'). The zone has been delegated (via an NS Resource Record) to this DNS.

The term 'master' was introduced in BIND 8.x and replaced the term 'primary'.

Master status is defined in BIND by including 'type master' in the zone declaration section of the named.conf file) as shown by the following fragment.

```
// example.com fragment from named.conf
// defines this server as a zone master
zone "example.com" in{
        type master;
        file "pri.example.com";
};
```

Notes:

1. The terms Primary and Secondary DNS entries in Windows TCP/IP network properties mean nothing, they may reflect the 'master' and 'slave' name-server or they may not - you decide this based on operational need, not BIND configuration.
2. It is important to understand that a zone 'master' is a server which gets its zone data from a local source as opposed to a 'slave' which gets its zone data from an external (networked) source (the 'master'). This apparently trivial point means that you can have any number of 'master' servers for any zone if it makes operational sense. You have to ensure (by a manual or other process) that the zone files are synchronised but apart from this there is nothing to prevent it.
3. Just to confuse things still further you may run across the term 'Primary Master' this has a special meaning in the context of dynamic DNS updates and is defined to be the name server that appears in the SOA RR record.

When a master DNS receives Queries for a zone for which it is authoritative then it will respond as 'Authoritative' (AA bit is set in a query response).

When a DNS server receives a query for a zone which it is neither a Master nor a Slave then it will act as configured (in BIND this behaviour is defined in the named.conf file):

1. If caching behaviour is permitted and recursive queries are allowed the server will completely answer the request or return an error.
2. If caching behaviour is permitted and Iterative (non-recursive) queries are allowed the server can respond with the complete answer (if it is already in the cache because of another request), a referral or return an error.
3. If caching behaviour NOT permitted (an 'Authoritative Only' DNS server) the server will return a referral or return an error.

A master DNS server can export (NOTIFY) zone changes to defined (typically slave) servers. This ensures zone changes are rapidly propagated to the slaves (interrupt driven) rather than rely on the slave server polling for changes. The BIND default is to notify the servers defined in NS records for the zone.

If you are running Stealth Servers and wish them to be notified you will have to add an also-notify parameter as shown in the BIND named.conf file fragment below:

```
// example.com fragment from named.conf
```

```
// defines this server as a zone master
// 192.168.0.2 is a stealth server NOT listed in a NS record
zone "example.com" in{
        type master;
        also-notify {192.168.0.2;};
        file "pri/pri.example.com";
};
```

You can turn off all NOTIFY operations by specifying 'notify no' in the zone declaration.

Example configuration files for a master DNS are provided.

4.2 Slave (Secondary) Name Servers

A Slave DNS gets its zone file information from a zone master and it will respond as authoritative for those zones for which it is defined to be a 'slave' and for which it has a currently valid zone configuration.

The term 'slave' was introduced in BIND 8.x and replaced the term 'secondary'.

Slave status is defined in BIND by including 'type slave' in the zone declaration section of the named.conf file) as shown by the following fragment.

```
// example.com fragment from named.conf
// defines this server as a zone slave
zone "example.com" in{
        type slave;
        file "sec/sec.example.com";
        masters {192.168.23.17;};
};
```

Notes:

1. The master DNS for each zone is defined in the 'masters' zone section and allows slaves to refresh their zone record when the 'expiry' parameter of the SOA Record is reached. If a slave cannot reach the master DNS when the 'expiry' time has been reached it will stop responding to requests for the zone. It will NOT use time-expired data.
2. The file parameter is optional and allows the slave to write the transferred zone to disc and hence if BIND is restarted before the 'expiry' time the server will use the saved data. In large DNS systems this can save a considerable amount of network traffic.

Assuming NOTIFY is allowed in the master DNS for the zone (the default behaviour) then zone changes are propagated to all the slave servers defined with NS Records in the master zone file. There can be any number of slave DNS's for any given 'master' zone. The NOTIFY process is open to abuse. BIND's default behaviour is to only allow

NOTIFY from the 'master' DNS. Other acceptable NOTIFY sources can be defined using the allow-notify parameter in named.conf.

Example configuration files for a slave DNS are provided.

4.3 Caching Name Servers

A Caching Server obtains information from another server (a Zone Master) in response to a host query and then saves (caches) the data locally. On a second or subsequent request for the same data the Caching Server will respond with its locally stored data (the cache) until the time-to-live (TTL) value of the response expires at which time the server will refresh the data from the zone master.

If the caching server obtains its data directly from a zone master it will respond as 'authoritative', if the data is supplied from its cache the response is 'non-authoritative'.

The default BIND behaviour is to cache and this is associated with the recursion parameter (the default is 'recursion yes'). There are many configuration examples which show caching behaviour being defined using a 'type hint' statement in a zone declaration. These configurations confuse two distinct but related functions. If a server is going to provide caching services then it must provide recursive queries and recursive queries need access to the root servers which is provided via the 'type hint' statement. A caching server will typically have a named.conf file which includes the following fragment:

```
// options section fragment of named.conf
// recursion yes is the default and may be omitted
options {
        directory "/var/named";
        version "not currently available";
        recursion yes;
};
// zone section
....
// the DOT indicates the root domain = all domains
zone "." IN {
        type hint;
        file "root.servers";
};
```

Notes:

1. BIND defaults to recursive queries which by definition provides caching behaviour. The named.conf recursion parameter controls this behaviour.
2. The zone '.' is shorthand for the root domain which translates to 'any domain not defined as either a master or slave in this named.conf file'.
3. cache data is discarded when BIND is restarted.

The most common DNS server caching configurations are:

- A DNS server acting as master or slave for one or more zones (domains) and as cache server for all other requests. A general purpose DNS server.
- A caching only local server - typically used to minimise external access or to compensate for slow external links. This is sometimes called a Proxy server though we prefer to associate the term with a Forwarding server

To cache or not is a crucial question in the world of DNS. BIND is regarded as the reference implementation of the DNS specification. As such it provides excellent - if complex to configure - functionality. The down side of generality is suboptimal performance on any single function - in particular caching involves a non-trivial performance overhead.

For general usage the breadth of BIND functionality typically offsets any performance concerns. However if the DNS is being 'hit' thousands of times per second performance is a major factor. There are now a number of alternate Open Source DNS servers some of which stress performance. These servers typically do NOT provide caching services (they are said to be 'Authoritative only' servers).

Example configuration files for a caching DNS are provided.

Note: The response to a query is Authoritative under three conditions:

1. The response is received from a Zone master.
2. The response is received from a Zone slave with non time-expired zone data.
3. The response is received by a caching server directly from either a Zone master or slave. If the response is read from the cache directly it is not authoritative.

4.4 Forwarding (a.k.a Proxy) Name Servers

A forwarding (a.k.a. Proxy, Client, Remote) server is one which simply forwards all requests to another DNS and caches the results. On its face this look a pretty pointless exercise. However a forwarding DNS sever can pay-off in two ways where access to an external network is slow or expensive:

1. Local DNS server caching - reduces external access and both speeds up responses and removes unnecessary traffic.
2. Remote DNS server provides recursive query support - reduction in traffic across the link - results in a single query across the network.

Forwarding servers also can be used to ease the burden of local administration by providing a single point at which changes to remote name servers may be managed, rather than having to update all hosts.

Forwarding can also be used as part of a Split Server configuration for perimeter defence.

BIND allows configuration of forwarding using the forward and forwarders parameters either at a 'global' level (in an options section) or on a per-zone basis in a zone section of the named.conf file. Both configurations are shown in the examples below:

Global Forwarding - All Requests

```
// options section fragment of named.conf
// forwarders can have multiple choices
options {
        directory "/var/named";
        version "not currently available";
        forwarders {10.0.0.1; 10.0.0.2;};
        forward only;
};
// zone file sections
....
```

Per Domain Forwarding

```
// zone section fragment of named.conf
zone "example.com" IN {
        type forward;
        file "fwd.example.com";
        forwarders {10.0.0.1; 10.0.0.2;};
};
```

Where dial-up links are used with DNS forwarding servers BIND's general purpose nature and strict standards adherence may not make it an optimal solution. A number of the Alternate DNS solutions specifically target support for such links. BIND provides two parameters dialup and heartbeat-interval (neither of which is currently supported by BIND 9) as well as a number of others which can be used to minimise connection time.

Example configuration files for a forwarding DNS are provided.

4.5 Stealth (a.k.a. DMZ or Split) Name Server

A stealth server is defined as being a name server which does not appear in any **publicly visible** NS Records for the domain. The stealth server is normally used in a configuration called Split Severs which can be roughly defined as having the following characteristics:

1. The organisation needs a public DNS to enable access to its public services e.g. web, mail ftp etc..

2. The organisation does not want the world to see any of its internal hosts either by interrogation (query or zone transfer) or should the DNS service be compromised.

A Split Server configuration is shown in Figure 4.1.

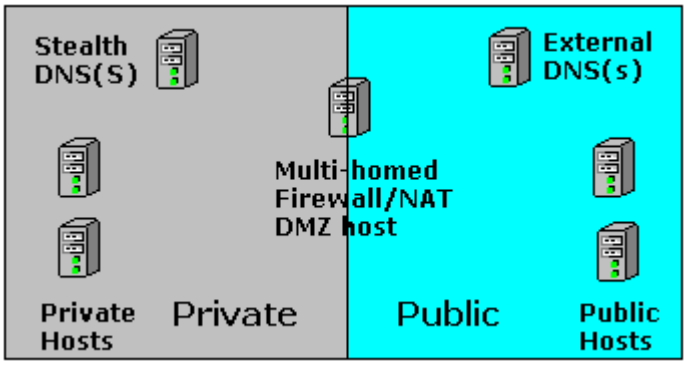


Figure 4.1 Split Server configuration

The external server(s) is(are) configured to provide Authoritative Only responses and no caching (no recursive queries accepted). The zone file for this server would be unique and would contain ONLY those systems or services that are publicly visible e.g. SOA, NS records for the public (not stealth) name servers, MX record(s) for mail servers and www and ftp service A records. Zone transfers can be allowed between between the public servers as required but they MUST NOT transfer or accept transfers from the Stealth server. While this may seem to create more work, the concern is that should the host running the external service be compromised then inspection of the named.conf or zone files must provide no more information than is already publically visible. If 'master', 'allow-notify','allow-transfer' options are present in named.conf (each of which will contain a private IP) then the attacker has gained more knowledge about the organisation - they have penetrated the 'veil of privacy'.

There are a number of articles which suggest that the view statement may be used to provide similar functionality using a single server but this does not address the problem of the DNS host system being compromised and by simple inspection of the named.conf file additional data about the organisation could be discovered. In our opinion 'view' does not provide adequate security in a 'Split DNS' solution.

A minimal public zone file is shown below:

```
; public zone master file
; provides minimal public visibility of external services
example.com.  IN      SOA   ns.example.com. root.example.com. (
                              2003080800 ; se = serial number
                              3h         ; ref = refresh
                              15m        ; ret = update retry
                              3w         ; ex = expiry
                              3h      ; min = minimum
                              )
```

```
                IN      NS      ns1.example.com.
                IN      NS      ns2.example.com.
                IN      MX  10  mail.example.com.
ns1             IN      A       192.168.254.1
ns2             IN      A       192.168.254.2
mail            IN      A       192.168.254.3
www             IN      A       192.168.254.4
ftp             IN      A       192.168.254.5
```

The internal server (the Stealth Server) can be configured to make visible internal and external services, provide recursive queries and all manner of other services. This server would use a private zone master file which could look like this:

```
; private zone master file used by stealth server(s)
; provides public and private services and hosts
example.com.  IN      SOA   ns.example.com. root.example.com. (
                              2003080800 ; se = serial number
                              3h         ; ref = refresh
                              15m        ; ret = update retry
                              3w         ; ex = expiry
                              3h         ; min = minimum
                              )
              IN      NS      ns1.example.com.
              IN      NS      ns2.example.com.
              IN      MX  10  mail.example.com.
; public hosts
ns1           IN      A       192.168.254.1
ns2           IN      A       192.168.254.2
mail          IN      A       192.168.254.3
www           IN      A       192.168.254.4
ftp           IN      A       192.168.254.5
; private hosts
joe           IN      A       192.168.254.6
bill          IN      A       192.168.254.7
fred          IN      A       192.168.254.8
....
accounting    IN      A       192.168.254.28
payroll       IN      A       192.168.254.29
```

Using BIND 9's view statement can provide different services to internal and external requests can reduce further the Stealth server's visibility e.g. forwarding all DNS internal requests to the external server.

Example configuration files for a stealth DNS are provided.

4.6 Authoritative Only Server

The term **Authoritative Only** is normally used to describe two concepts:

1. The server will deliver Authoritative Responses - it is a zone master or slave for one or more domains.
2. The server will NOT cache.

There are two configurations in which Authoritative Only servers are typically used:

1. As the public or external server in a Split (a.k.a. DMZ or Stealth) DNS used to provide perimeter security.
2. High Performance DNS servers. In this context general purpose DNS servers such as BIND may not provide an ideal solution and there are a number of Open Source Alternatives some of which specialise in high performance Authoritative only solutions.

You cannot completely turn off caching in BIND but you can control it and provide the functionality described above by simply turning off recursion in the 'option' section of named.conf as shown in the example below.

```
// options section fragment of named.conf
// recursion no = limits caching
options {
        directory "/var/named";
        version "not currently available";
        recursion no;
};
// zone file sections
....
```

BIND provides three more parameters to control caching ,max-cache-size and max-cache-ttl neither of which will have much effect on performance in this particular case and allow-recursion which uses a list of hosts that are permitted to use recursion (all others are not).

Example configuration files for a authoritative-only DNS are provided.

Chapter 5. BIND (Berkeley Internet Name Daemon)

This chapter describes HOWTO install BIND 9.3.0 on a variety of OS Platforms as well as BIND's command line arguments. Finally - OK so everyone knows this but we didn't the first time we touched BIND (yeah we know it shows) - BIND runs as the daemon **named**.

- FreeBSD Install (4.x and 5.x)
- Linux Install (Fedora Core 2)
- Windows Install (Win2K and NT 4.0)
- BIND Command Line Arguments

FreeBSD Installation

FreeBSD 4.x and 5.1 ships with BIND version 8.x as the default or base installation. FreeBSD 5.3 - the first of the stable 5.x series - ships with Bind 9.3.0 and some annoying traits.

FreeBSD differentiates between a base DNS install and a normal DNS install. There are some serious choices to be made when installing from the ports system. We assume the theory behind this is to enable experimentation with the new software but with the ability to return to the original DNS software by changing configuration options in the rc.conf file if things get a bit wobbly.

You can either install BIND 9 *as well as* the default BIND 8 or 9 installation or you can replace the base version. The difference is the base version is installed in /usr/sbin (and the tools in /usr/bin) whereas a normal (non-base) installation is made to /usr/local/sbin (and the tools to /usr/local/bin). Finally the standard version assumes the named.conf file in /etc/namedb/named.conf whereas a non-base install assumes /usr/local/etc/named.conf.

Notes:

1. On our very *dirty* FreeBSD 4.x test system - we have done a lot of very naughty things to this poor system none of them deliberately! - we failed to get Bind9 to install initially. The DNS make kept failing with undefined's during compilations in exotic modules caused by incompletely generated header files - created by the *gen* program which is in turn built during the install! We eventually tracked the the problem to an install of /usr/local/lib/libnsl which was causing the installation to assume a linux base and hence the *gen* program failed to generate the headers correctly. We deleted the library (it's not normally installed in FreeBSD) re-ran and all was well again. The Bind9 build process is really rather horrible we have concluded. Maybe its all essential but complex man, complex. Still it forced us to find out more about autoconf and configure and automake and make and gmake... We had to take a couple of days off work to recover.

BIND 9 non-base install

Assuming you have updated the ports-dns collection proceed as normal:

```
cd /usr/ports/dns/bind9
make install clean
```

The above sequence installs BIND9 in /usr/local/sbin and the tools in /usr/local/bin and assumes the named.conf file is in /usr/local/etc.

If you want to run BIND9 at startup you must edit /etc/rc.conf as follows:

```
# add following line if not present
named_enable="YES"
# the line below must replace the line named_program="/usr/sbin/named'
if present
# otherwise add it
named_program="/usr/local/sbin/named"
```

Either copy your named.conf file from /etc/namedb to /usr/local/etc before you restart Bind or create a new version of the file in this directory.

To use the BIND9 tools you must precede the command with the BIND9 tool directory path e.g.

```
# this will run the installed Bind9 version
/usr/local/bin/dig example.com any
# but this command
dig example.com
# this will run the base dig version
# 4.x and 5.1 = Bind 8, 5.3 = Bind 9.3.0
dig example.com
```

BIND 9 replace base install

This assumes you either want to run the latest version of BIND as the base system - replacing the exiting BIND - or a new install with Bind 9 as the base system. Assuming you have updated the ports-dns collection proceed as follows:

```
cd /usr/ports/dns/bind9
make PORT_REPLACES_BASE_BIND9=yes install clean
```

The above sequence installs BIND9 in /usr/sbin and the tools in /usr/bin.

If you want to run BIND9 at startup you may need to edit /etc/rc.conf as follows:

```
# add following line if not present
named_enable="YES"
# add the line following line if not present
named_program="/usr/sbin/named"
```

No special action is required to run BIND9 tools:

```
# this will run the Bind9 dig version
dig example.com
```

FreeBSD 5.3 Issues

By default FreeBSD 5.3 installs Bind 9 (9.3.0) as the default (or base) version but with the following wrinkles:

1. Bind9 defaults to run in a chroot jail or a sandbox (which nows seems the "in" term for a jail) in which *all* BIND9 files are maintained under /var/named - including named.conf, log files and pid files (hard links are provided so you can continue to find the files where you thought they would be). To disable the sandbox add to /etc/rc.conf the following line(s):

```
2. named_chrootdir=""               # disables jail/sandbox
3. named_pidfile="/var/run/named/pid" # Must set this in named.conf as well
```

```
4. named_chroot_autoupdate="NO"       # Automatically install/update chrooted
5.                                    # components of named. See /etc/rc.d/named.
6. named_symlink_enable="NO"            # Symlink the chrooted pid file
```

The default value of these parameters in /etc/defaults/rc.conf are:

```
named_enable="YES"                  # Run named, the DNS server (or NO).
named_program="/usr/sbin/named"          # path to named, if you want a
different one.
named_flags="-u bind"               # Flags for named
named_pidfile="/var/run/named/pid" # Must set this in named.conf as well
named_chrootdir="/var/named"        # Chroot directory (or "" not to auto-chroot
it)
named_chroot_autoupdate="YES"       # Automatically install/update chrooted
                                    # components of named. See /etc/rc.d/named.
named_symlink_enable="YES"          # Symlink the chrooted pid file
```

As always you should not update the /etc/defaults/rc.conf file but rather edit /etc/rc.conf which will replace entries already defined in /etc/defaults/rc.conf.

7. The system does not ship with localhost or localhost.ca (localhost.rev in our terminology) files instead there is a script /etc/namedb/named.sh which will help you define these files.

Fedora Core 2 Installation

This section describes the installation of BIND 9.3.0 on a clean Fedora Core 2 system. This was the first Linux system we had used used since Redhat 7.x series and we found it significantly more windows like (this may be good or bad depending on your predilictions for these things). It was also the first time we had ever had a X11 graphical installation work without fooling around with XF86Config config files (or the /etc/X11/xorg.conf as it is now). How much of this is a function of time and better probing routines and how much the change to Xorg we have no real idea. Never-the-less if you want a graphical system it surely was real fun (read difficult) to configure X-Windows in the old days and it appears at least one of the objections to Linux vs Windows has been removed if you are into GUIs. The configuration had the following highlights:

1. We used the graphical configuration option (anaconda)
2. Automatic Partitioning was selected
3. A server install was selected
4. We selected a GNOME based desktop - we had read somewhere that RH had elected to go with GNOME (rather than KDE) - and we were curious to see what stage the GUI stuff had reached. We would not normally install a GUI for a server configuration.
5. Since we were going to install BIND 9.3.0 we did not select to install DNS from the installation menu. In fact, as we shall see, it partially installed BIND-9.2.8-13.
6. We selected very few install options other that the development tools and graphical internet utilities - this later to allow us to get a hold of the

various RPMs and tarballs. We would use the browser for FTPing the rpms and the former for use with the various tarballs.

Installation was very uneventful and we were up in < 30 minutes. **Note:** We were using a VIA M10000 motherboard (we love these systems) based server and had failed to install FC3 on this system so we retreated fairly quickly to FC2. We note in passing that Fedora is pushing releases out a pretty quick rate these days. We also noted that FC2 does not support the latest SE Linux options - well it does but they are disabled 'cos they don't work. You need at least FC3 for this functionality or even FC4 which is on the horizon.

BIND 9 Install

We used mozilla (the FC2 default installed browser) to obtain, from rpmfind.net, Fedora Core Development rpms for:

1. bind-9.3.0-2.i386.rpm
2. bind-utils-9.3.0-2.i386.rpm
3. bind-libs-9.3.0-2.i386.rpm
4. bind-devel-9.3.0-2.i386.rpm

The above were all downloaded to /tmp. Before we installed we ran the following from a terminal window:

```
rpm -q bind
# which returned
bind-9.2.8-13
```

In retrospect this was surprising (since we had not requested any DNS software during the FC2 installation procedure - FC2 install was kind enough to do it anyway) and perhaps we should have removed this version before installing since we ended up with -in our view - an incomplete installation. We may have had a complete install if we had done this - maybe not. We issued the following command to upgrade the installation:

```
rpm -Uvh /tmp/bind*
```

every thing appeared to work - there were no error messages. So far so good.

BIND 9 Configure

So we had a bright shiny FC2 GNOME destktop with BIND 9.3.0 installed in front of us and now to get BIND up and running. We were as conversant as normal with the details of any system that was new to us - we read nothing but figured we could wing it and hit google if we got into trouble. We rounded up the normal suspects looking for the default named.conf and default zone files in the normal Linux locations being /etc/named.conf and /var/named for the zone files - typically named.ca for the root servers - and localhost.zone and named.local (one day someone may explain the the idea behind the naming scheme here - we suspect

drunken revelry ourselves). We discovered only a file called named.custom in /etc which was a vestigial named.conf file. We poked around a bit more and then found a GUI BIND configuration tool under System Settings-> Server Settings-> DNS from the main menu (which is very windows like). After 5 minutes and a look at the help file for system-config-bind we abandoned the GUI method. Why it could not provide a set of default configurations to include caching server configuration (or at the very least default local host files) is beyond us. What is the point in having a GUI utility that forces you to know just as much as if you manually configure - and the help files don't even hyperlink the BIND documentation - beats us. This GUI for the sake of GUI which - IOHO - misses the point about the power of a GUI.

We manually built a simple caching server named.conf (no we didn't use vi - we used gEdit). We FTP'd the missing root.server(called named.ca in RH Lunix distributions - also available here as named.root) and master.localhost (called localhost.zone in RH Lunix distributions) and localhost.rev (called named.local in RH Lunix distributions) from another system - if you can't do this you will have to manually create them. We always log to /var/log/named/named.txt (habit really) so we had to create and fix the permissions for this directory using:

```
cd /var/log
mkdir named
chown named:named named
```

We then fired up BIND using:

```
/etc/rc.d/init.d/named start
```

And it failed with a segmentation fault!

syslog (/var/log/messages) showed the following message:

```
/usr/lib/libisc.so.9: symbol __snprint_chk, version GLIBC_2.3.4
not defined in file libc.so.6 with link time reference failed
```

We checked the version of GLIBC:

```
rpm -q glibc
glibc-2.3.3-27
```

Surprising that rpm forgot the dependency on glibc!! We downloaded the following Fedora Core Development rpms - all demanded by various dependencies:

1. glibc-2.3.4-3
2. glibc-common-2.3.4-3
3. glibc-devel-2.3.4-3
4. glibc-headers-2.3.4-3
5. nscd-2.3.4-3
6. selinux-1.20.1-2

The easiest way we know to solve all those terrible dependency with rpms is to create a new folder (in this case /tmp/glibc) - download or move all the rpms into it and then:

```
rpm -Uvh /tmp/glibc/*
glibc-2.3.3-27
```

We fired up bind again:

```
/etc/rc.d/init.d/named start
```

and confirmed it was running:

```
ps ax |grep named
1846   ?    S    0:00 /usr/sbin/named -u named
```

Finally we verified that all was working using a dig:

```
dig @192.168.2.2 example.com any
# where @192.168.2.2 is the new server address
# forces use of new server irrespective of
# resolv.conf configuration
```

Which returned a good result and confirmed that all was working. We wanted this server to use our new DNS so we edited /etc/resolve.conf to include this servers IP as the first **namesever** record.

. We also want bind to load at start-up time so we need to ensure the named script in invoked - the named script was installed as usual as /etc/rc.d/init.d/named by either the initial install or the upgrade - who knows). To do this we linked the named start/stop script for each of the run-level directories that we might use:

```
ln /etc/rc.d/init.d/named /etc/rc.d/rc5.d/S68named # gui interface
ln /etc/rc.d/init.d/named /etc/rc.d/rc5.d/K68named # gui interface
ln /etc/rc.d/init.d/named /etc/rc.d/rc3.d/S68named # tty interface
ln /etc/rc.d/init.d/named /etc/rc.d/rc3.d/K68named # tty interface
```

The 68 in the S and K values is pretty arbitrary.

Finally we rebooted the system, checked syslog for any errors and verified that BIND was indeed running, again with the command:

```
ps aux |grep named
1846   ?    S    0:00 /usr/sbin/named -u named
```

In our follow-up research we discovered we had two other choices for installation as well as the standard rpms:

1. caching-nameserver rpm which seems to be a package installed after BIND has been installed.
2. bind-chroot rpms

We could find no further documentation that describes use, specification of installation instructions for either of these packages. We're sure it must exist.

BIND Command Line Arguments

This section describes the various command line options for BIND. You can get these using **man named** but reproduced here for consistency.

Arg	Param	Notes
-c	/path/to/config-file	Absolute path to the config file (named.conf). This allows you to both change the location and the name of this file. Default depends of OS (Linux = /etc/named.conf, BSD = either /etc/namedb/named.conf or /etc/local/etc/named.conf, Windows = c:\winnt\system32\dns\etc\named.conf.
-d	de-bug level	
-f	-	run in foreground (don't run as daemon) - normally only for de-bug purposes.
-g	-	run in foreground (don't run as daemon) and log to stderr (console) - normally only for de-bug purposes.
-n	#cpus	Create #cpus worker threads to take advantage of multiple CPUs. If not specified, named will try to determine the number of CPUs present and create one thread per CPU. If it is unable to determine the number of CPUs, a single worker thread will be created.
-p	port-no	Listen on defined port number. Default is 53. Normally only used for debugging purposes since queries are received on port 53.
-t	directory	The path to the a directory to be used when named is run in a sandbox (a chroot jail). This is conventionally set to /var/named/chroot in most systems which offer this service as a standard configuration or option (BSD and FC2 do) but can be set to anything you want. Must be used in conjunction with the -u argument below to provide any meaningful security.
-u	user	Cause bind to **suid()** (change user name) after creating sockets on port 53 (which is in the privileged range of < 1024). If not present runs as user **root**. Generally used only with **chroot** options (-t above) but most start-up scripts now use -u named argument even if not chrooted which means that log files files will have to have appropriate permissions set.
-v	-	Displays the bind version number to stdout (console) and exit.

There are two further arguments (-s and -x) which should only be used by developers and have been omitted.

6. DNS Sample BIND Configurations

This chapter provides a number of BIND configuration samples.

6.1 Sample BIND Configuration Overview

This chapter provides sample configurations and descriptions for each of the DNS types previously described. A BIND systems consists of the following parts:

1. A named.conf file describing the functionality of the BIND system. The entries in this file are fully described.
2. Depending on the configuration one or more zone files describing the domains being managed. The entries in zone files are fully described. Zone files contain Resource Records which are fully described.
3. Depending on the configuration one or more required zone files describing the 'localhost' and root name servers.

Many BIND/DNS configurations are schizophrenic in nature - they may be 'masters' for some zones, 'slaves' for others, forward others and provide caching services for all comers. Where possible we cover alternate configurations or as least note the alternate configurations.

All the configuration files are deliberately kept simple - links are provided to the various sections that will describe more 'advanced' parameters as appropriate. Comments are included in the files to describe functionality. The configuration used throughout is:

1. Two name servers are used one internal (ns1) and one external (ns2) to the domain
2. The mail service is external to the domain (provided by a third party)
3. FTP and WWW services are provided by the same host
4. There are two hosts named bill and fred
5. The host address are all in the class C private address range 192.168.0.0 (a slightly artificial case)

6.1.1 Zone File Naming Convention

Everyone has their own ideas on a good naming convention and thus something that is supposed to be useful becomes contentious.

Here is a convention that is in daily use. Its sole merits are that it is a convention and makes sense to its authors.

1. All zone files are placed in /var/named/. The base directory contains all the housekeeping zone files (e.g. localhost, reverse-mapping, root.servers etc.) with a subdirectory structure used as follows:

    1. /var/named/master - master zone files
    2. /var/named/slave - slave zones files
    3. /var/named/views - where views are used
2. master files are named master.example.com (or master.example.net etc.) if its a sub-domain it will be master.sub-domain.example.com etc.
3. slave zone files are named slave.example.com (or slave.example.ca etc.) if its a sub-domain it will be slave.sub-domain.example.com etc.
4. The root server zone file is called root.servers (typically called named.ca or named.root in BIND distributions).
5. The reverse mapping file name uses the subnet number and .rev i.e.. if the zone is '23.168.192.IN-ADDR.ARPA' the file is called 192.168.23.rev to save having to reverse the digits at 3AM in a blind panic.
6. The 'localhost' zone file is called master.localhost (typically called localhost.zone on BIND distributions). The reverse mapping file is called localhost.rev (typically called named.local in BIND distributions).

Note: For most Linux distributions you have a small overhead at the beginning to rename the supplied files but the author considers it worthwhile in the long run to avoid confusion.

Final point on this topic: Whatever your convention be rigorous in its application!

6.2 Master (Primary) DNS Server

The functionality of the master name server was previously described.

Master Name Server Configuration

The BIND DNS configuration provides the following functionality:

1. 'master' DNS for example.com
2. provides 'caching' services for all other domains
3. provides recursive query services for all resolvers

The BIND 'named.conf' is as follows (click to look at any file):

```
// MASTER & CACHING NAME SERVER for EXAMPLE, INC.
// maintained by: me myself alone
// CHANGELOG:
// 1. 9 july 2003 - did something
// 2. 16 july 2003 - did something else
// 3. 23 july 2003 - did something more
//
options {
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "get lost";
  // optional - disables all transfers - slaves allowed in zone clauses
  allow-transfer {"none";);
};
//
```

```
// log to /var/log/named/example.log all events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// BIND 8.x logging MUST COME FIRST in this file
// BIND 9.x parses the whole file before using the log
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 2m;
   severity info;
 };
 category default{
  example_log;
 };
};
// required zone for recursive queries
zone "." {
  type hint;
  file "root.servers";
};
zone "example.com" in{
  type master;
  file "master/master.example.com";
  // enable slaves only
         allow-transfer {192.168.23.1;192.168.23.2;);
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
// reverse map for class C 192.168.0.0
zone "0.168.192.IN-ADDR.ARPA" in{
  type master;
  file "192.168.0.rev";
};
```

Notes:

6.3 Slave (Secondary) DNS Server

The functionality of the slave name server was previously described.

Slave Name Server Configuration

The BIND DNS configuration provides the following functionality:

1. 'slave' DNS for example.com
2. provides 'caching' services for all other domains

3. provides recursive query services for all resolvers

Note: Since we are defining the slave the alternate sample file is used throughout this example configuration with all servers being internal to the domain.

The BIND 'named.conf' is as follows (click to look at any file):

```
// SLAVE & CACHING NAME SERVER for EXAMPLE, INC.
// maintained by: me myself alone
// CHANGELOG:
// 1. 9 july 2003 - did something
// 2. 16 july 2003 - did something else
// 3. 23 july 2003 - did something more
//
options {
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
  // allows notifies only from master
  allow-notify {192.168.0.1};
  // disables all zone transfer requests
  allow-transfer{"none"};
};
//
// log to /var/log//named/example.log all events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// BIND 8.x logging MUST COME FIRST in this file
// BIND 9.x parses the whole file before using the log
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
  file "/var/log/named/example.log" versions 3 size 2m;
  severity info;
 };
 category default{
  example_log;
 };
};
// required zone for recursive queries
zone "." {
  type hint;
  file "root.servers";
};
// see notes below
zone "example.com" in{
  type slave;
  file "slave/slave.example.com";
  masters {192.168.0.1;};
};
// required local host domain
zone "localhost" in{
  type master;
  file "pri.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
// reverse map for class C 192.168.0.0 (see notes)
zone "0.168.192.IN-ADDR.ARPA" IN {
  type slave;
  file "sec.192.168.0.rev";
```

```
  masters {192.168.0.1;};
};
```

Notes:

1. The slave zone file 'slave/slave.example.com' is optional and allows storage of the current records - minimising load when named is restarted. To create this file initially just open and save an empty file. BIND will complain the first time it loads but not thereafter.
2. The reverse map for the network (zone 0.168.192.IN-ADDR.ARPA) is defined as a slave for administrative convenience - you need to maintain only one copy - but it could be defined as a 'master' with a standard reverse map format.
3. A single 'masters' IP address is used specifying ns1.example.com.

6.4 Caching Only DNS Server

The functionality of the Caching Only name server was previously described.

Caching Only Name Server Configuration

The BIND DNS configuration provides the following functionality:

1. The name server is not a 'master' or 'slave' for any domain
2. provides 'caching' services for all domains
3. provides recursive query services for all resolvers

The BIND 'named.conf' is as follows (click to look at any file):

```
// CACHING NAME SERVER for EXAMPLE, INC.
// maintained by: me myself alone
// CHANGELOG:
// 1. 9 july 2003 - did something
// 2. 16 july 2003 - did something else
// 3. 23 july 2003 - did something more
//
options {
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
  // disables all zone transfer requests
  allow-transfer{"none"};
};
//
// log to /var/log/example.log all events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// BIND 8.x logging MUST COME FIRST in this file
// BIND 9.x parses the whole file before using the log
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 2m;
   severity info;
```

```
 };
 category default{
  example_log;
 };
};
// required zone for recursive queries
zone "." {
  type hint;
  file "root.servers";
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
```

Notes:

1. The Caching only name server contains no zones (other than 'localhost') with 'master' or 'slave' types.
2. The reverse map zone has been omitted since it assumed that an external body (ISP etc) has the master domain DNS and is therefore also responsible for the reverse map. It could be added if required for local operational reasons.

6.5 Forwarding (a.k.a. Proxy, Client, Remote) DNS Server

The functionality of the Forwarding name server was previously described.

Forwarding Name Server Configuration

The BIND DNS configuration provides the following functionality:

1. The name server is not a 'master' or 'slave' for any domain
2. provides 'caching' services for all domains
3. forwards all queries to a remote DNS from all local resolvers (Global forwarding)

The BIND 'named.conf' is as follows (click to look at any file):

```
// FORWARDING & CACHING NAME SERVER for EXAMPLE, INC.
// maintained by: me myself alone
// CHANGELOG:
// 1. 9 july 2003 - did something
// 2. 16 july 2003 - did something else
// 3. 23 july 2003 - did something more
```

```
//
options {
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
  forwarders {10.0.0.1; 10.0.0.2;};
  forward only;
  // disables all zone transfer requests
  allow-transfer{"none"};
};
// log to /var/log/example.log all events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// BIND 8.x logging MUST COME FIRST in this file
// BIND 9.x parses the whole file before using the log
// failure messages up to this point are in (syslog) /var/log/messages
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3;
   severity info;
 };
 category default{
  example_log;
 };
};
// required local host domain
zone "localhost" in{
  type master;
  file "pri.localhost";
  allow-update{none;};
};
// localhost reverse map
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
```

Notes:

1. The Forwarding name server typically contains no zones (other than 'localhost') with 'master' or 'slave' types.
2. The reverse map zone has been omitted since it assumed that an external body (ISP etc) has the master domain DNS and is therefore also responsible for the reverse map. It could be added if required for local operational reasons.
3. The forward option must be used in conjunction with a forwarders option. The value 'only' will override 'recursive query' behaviour.
4. Since all queries are forwarded the root servers zone ('type hint') can be omitted.
5. Forwarding can be done on a zone basis in which case the values defined override the global options.

6.6 Stealth (a.k.a. Split or DMZ) DNS Server

The functionality of the Stealth name server was previously described. The following diagram illustrates the conceptual view of a Stealth (a.k.a. Split) DNS server system.

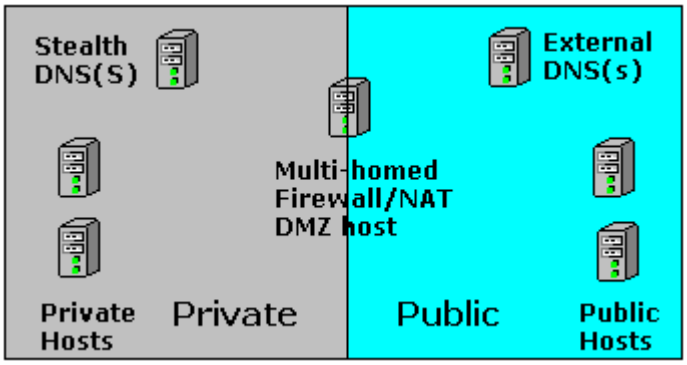


Figure 6.1 Split/Stealth Server configuration

The key issue in a 'Stealth' (a.k.a. Split) DNS system is that there is a clear line of demarcation between the 'Internal' Stealth server(s) and the 'External' or Public DNS servers(s). The primary difference in configuration is the 'Stealth' Servers will provide a comprehensive set of services to internal users to include caching and recursive queries and would be configured as a typical Master DNS, while the External server may provide limited services and would typically be configured as an Authoritative Only DNS server.

There are two critical points:

1. The zone file for the 'Stealth' server will contain both public and private hosts, whereas the 'Public' server's master zone file will contain only public hosts.
2. To preserve the 'Stealth' nature it is vital that the PUBLIC DNS configuration does not include options such as 'master', 'allow-notify','allow-transfer', etc. with references to the IP of the 'Stealth' server. If the Stealth servers IP where to appear in the Public DNS server and its file system were to be compromised the attacker could gain more knowledge about the organisation - they can penetrated the 'veil of privacy' by simply inspecting the 'named.conf file.

There are a number of articles which suggest that the view statement may be used to provide similar functionality using a single server. This does not address the problem of the DNS host system being compromised and by simple 'named.conf' file inspection additional data about the organisation being discovered. In a secure environment 'view' does not provide a 'Stealth DNS' solution if there is any possibility that a filesystem compromise can happen.

6.7 Authoritative Only DNS Server

The functionality of the Authoritative name server was previously described. If security is not the primary requirement then the view statement may be used to

provide 'Authoritative only' services to external users and more comprehensive services to internal users. An example configuration is shown below.

Authoritative Only Name Server Configuration

The BIND DNS configuration provides the following functionality:

1. 'master' DNS for example.com
2. does NOT provide 'caching' services for any other domains
3. does NOT provide recursive query services for all resolvers (Iterative only)
4. optimised for maximum performance

The BIND 'named.conf' is as follows (click to look at any file):

```
// AUTHORITATIVE ONLY NAME SERVER for EXAMPLE, INC.
// maintained by: me myself alone
// CHANGELOG:
// 1. 9 july 2003 - did something
// 2. 16 july 2003 - did something else
// 3. 23 july 2003 - did something more
//
options {
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
  version "not currently available";
  recursion no;
  // disables all zone transfer requests in this case
  // for performance not security reasons
  allow-transfer{none;};
};
//
// log to /var/log/example.log all events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// BIND 8.x logging MUST COME FIRST in this file
// BIND 9.x parses the whole file before using the log
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 2m;
   severity info;
 };
 category default{
  example_log;
 };
};
zone "example.com" in{
  type master;
  file "master/master.example.com";
};
// reverse map for class C 192.168.0.0
zone "0.168.192.IN-ADDR.ARPA" in{
  type master;
  file "192.168.0.rev";
};
// required local host domain
zone "localhost" in{
  type master;
  file "master.localhost";
  allow-update{none;};
};
```

```
// localhost reverse map
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "localhost.rev";
  allow-update{none;};
};
```

Notes:

1. The reverse mapping zone (zone "0.168.192.IN-ADDR.ARPA") was originally omitted from this server - given the use of reverse look-up by anti-spam systems we have restored the reverse look-up zone. It would - in a perfect world without spam - typically not be present on a performance oriented server.

BIND provides three more parameters to control caching ,max-cache-size and max-cache-ttl neither of which will have much effect on performance in the above case and allow-recursion which uses a list of hosts that are permitted to use recursion (all others are not) - a kind of poor man's 'view'.

6.8 View based Authoritative Only DNS Server

The functionality of the Authoritative name server was previously described. If security is not the primary requirement then the view statement may be used to provide 'Authoritative only' services to external users and more comprehensive services to internal users.

View based Authoritative Only Name Server Configuration

The BIND DNS configuration provides the following functionality:

1. 'master' DNS for example.com
2. does NOT provide 'caching' services for any external users
3. does NOT provide recursive query services for any external resolvers (Iterative only)
4. provides 'caching' services for internal users
5. provides recursive query services for internal users

The BIND 'named.conf' is as follows (click to look at any file):

```
// VIEW BASED AUTHORITATIVE ONLY NAME SERVER for EXAMPLE, INC.
// maintained by: me myself alone
// CHANGELOG:
// 1. 9 july 2003 - did something
// 2. 16 july 2003 - did something else
// 3. 23 july 2003 - did something more
//
// global options
options {
  directory "/var/named";
  // version statement for security to avoid hacking known weaknesses
```

```
  version "not currently available";
};
//
// log to /var/log/example.log all events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// BIND 8.x logging MUST COME FIRST in this file
// BIND 9.x parses the whole file before using the log
// failure messages up to this point are in (syslog) /var/log/messages
//
  logging{
  channel example_log{
   file "/var/log/named/example.log" versions 3 size 2m;
   severity info;
 };
 category default{
  example_log;
 };
};
// provide recursive queries and caching for our internal users
view "goodguys" {
 match-clients { 192.168.0.0/24; }; // our network
  recursion yes;
  // required zone for recursive queries
  zone "." {
   type hint;
   file "root.servers";
  };
  zone "example.com" {
   type master;
   // private zone file including local hosts
   file "view/master.example.com.internal";
  };
  // required local host domain
  zone "localhost" in{
   type master;
   file "master.localhost";
   allow-update{none;};
  };
  // localhost reverse map
  zone "0.0.127.in-addr.arpa" in{
   type master;
   file "localhost.rev";
   allow-update{none;};
  };
}; // end view

// external hosts view
view "badguys" {
 match-clients {"any"; }; // all other hosts
  // recursion not supported
  recursion no;
 zone "example.com" {
   type master;
          // only public hosts
   file "view/master.example.com.external";
  };
}; // end view
```

Notes:

1. All the required zones must be declared in each view.
2. The 'goodguys' view contains the root.servers, 'localhost' and reverse mapping file.

3. The 'badguys' view contains only the required zone files for which we will answer authoritatively.
4. The 'badguys' view may contain an edited version of the reverse map file.

7. 'named.conf' Parameters

This chapter describes the BIND 9.3.x **named.conf** file which controls the behaviour and functionality of BIND. **named.conf** is the only file which is used by BIND - confusingly there are still many references to **boot.conf** which was used by BIND 4 - ignore them.

BIND releases include a list of the latest statements and options supported. This list is available in /usr/share/docs/bind-version/misc/options (redhat) or /usr/src/contrib/bind/doc/ (FreeBSD) and if you are using the Windows version it ain't there! Supported list for BIND 9.3.0.

BIND allows a daunting list of configuration entities. You need a small subset to get operational. Read the first two sections to get a feel for the things you need, it identifies the MINIMAL values (depending on your requirement). Check the samples section for configuration specific examples.

Note: We got fed-up with inconsistent terminology so we use the term **clause** to describe the structure that group together a set of related **statements**. We don't call 'em options or clauses or substatements(!) or statements or phrases - just clauses and statements. Period. If you want to read more about our reasons. Full list of statements.

named.conf format, structure and overview
named.conf required zone files

The **clauses** supported by BIND are:

acl	Access Control Lists. Defines one of more access control lists, groups of hosts or users identified by keys, that may be referenced in view and other clauses or statements.
controls	Describes and controls access to the control channel used by the remote administrator when using the rndc utility.
include	Neither a statement nor a clause. Included here for no particularly good reason. **include** can appear anywhere in a named.conf file either inside or outside a clause. Allows inclusion of external files into named.conf for administrative convenience or security reasons.
key	Defines shared keys used to control and authenticate operations such as Dynamic DNS (DDNS) and the remote control channel (the controls clause).

logging	Configures the location, level and type of logging that BIND performs. Unless you are using **syslog** you need a logging statement for BIND.
lwres	Defines the properties of BIND when running as a lightweight resolver.
options	Groups statements that control generic or global behavior and that have scope for all zones and views unless overridden within a zone, views or other clause.
server	Defines the properties or behavior this server will use when accessing or responding to a defined remote server
trusted-keys	
view	Controls BIND functionality and behaviour based on the host address(es).
zone	Defines the specific zones that your name server will support. In addition there are a number of special zones that you may need to include.

Statements Classification

We have also classified all the statements into the following groups:

1. Queries - statements controlling query behavior
2. Transfer - statements controlling zone transfer and DDNS behavior
3. Operations - statements controlling operational behavior
4. Security - statements controlling security behavior
5. Statistics - statements controlling statistical logging behavior

A full list of all statements is here.

named.conf format, structure and overview

A **named.conf** file can contain comments and will contain a number of **clauses** which group together related **statements** which control the functionality and security of the BIND server.

BIND provides a number of comment formats as follows:

```
/* C style comment format needs opening and closing markers
** but allows multiple lines or */
/* single lines */
// C++ style comnents single line format no closing required
# PERL/SHELL style comments single lines no closing required
```

The whole **named.conf** file is parsed for completeness and correctness before use - this is a major change from previous releases of BIND. Prior to the availability of (or in the absence of) a valid logging clause failures use **syslogd** and are (depending on your syslog.conf file) typically written to /var/log/messages thereafter failures are written to any file(s) defined in your **logging** clause. There are some rules defined for the clause order for BIND 9. The general clause layout of a named.conf file is usually:

```
// acl clause if required
// defining first avoids forward name references
acl "name" {...};
logging {...};
// usually requires at least a file statement
// unless you are using the system log
options {...};
// other clauses/statements (as required)
// zones clauses including 'required' zones
zone {...};
....
zone {...};
```

If you are using view clause the order changes significantly:

```
// acl clauses if required
// defining first avoids forward name references
acl "name" {...};
logging {...}
// usually requires at least a file statement
// unless you are using the system log
options {...};
// other clauses/statements (as required)
view "first" {
 options{...};
       // zones clauses including 'required' zones
 zone {...};
 .....
 zone {...};
};
view "second" {
 options {...};
 // zones clauses including 'required' zones
 zone {...};
 .....
 zone {...};
};
```

BIND is very picky about opening and closing brackets/braces, semicolons and all the other separators defined in the formal 'grammars' below, you will see in the literature various ways to layout statements. These variations are simply attempts to

minimise the chance of errors, they have no other significance. Use the method you feel most comfortable with.

named.conf required zone files

Depending on your requirements BIND needs a number of zone files to allow it to function properly - these are in addition to any zones files that explicitly describe master or slave zones:

root.servers

This file (called named.ca or named.root in most distributions but renamed root.servers in this guide) defines a list of name servers (a.root-servers.net - m.root-servers.net) where BIND can get a list of TLD servers for the particular TLD e.g. .com - perhaps that's why its called hint. When a name server cannot resolve a query it uses the name server list obtained to provide a referral (if its an Iterative query) or to find an answer (if its a Recursive query). The root server file is defined using a normal zone clause with **type hint** as in the example below:

```
zone "." {
  type hint;
  file "root.servers";
};
```

The 'zone "."' is short for the *root* zone and means any zone for which there is no locally defined zone (slave or master) or cached answer.

By convention this file is usually included as the first zone statement but there is no good reason for this - it may be placed anywhere suitable. If you are running an internal name service on a closed network you do not need the root.servers file or 'hint' zone. Even if the hint zone is not defined BIND 9 has a internal list which it uses.

The file supplied with any distribution will get out of date and can be updated from a number of locations including ICANN. You see numerous commentators advise that this file be updated every three months or so. This is not essential. The first thing that BIND does when loaded with a 'hint' zone' is to update the root-server list from one of the locations in the root.server file. It will log any discrepancies from the supplied file but carry on using its retrieved list. Other than extra log messages there seems little advantage in updating the root.server file unless BIND load time is vital. If you are curious to see a sample root.server file.

localhost

This zone allows resolution of the name 'localhost' to the loopback address 127.0.0.1 when using the DNS server. Any query for 'localhost' from any host using the name server will return 127.0.0.1. **localhost** is used by many applications. On its face this may seem a little strange and you can either continue to treat the process as magic or get some understanding of how resolvers work. The localhost zone is defined as shown below

```
zone "localhost" in{
  type master;
  file "master.localhost";
};
```

In many examples and even the files supplied with BIND 9 a zone specific option allow-update statement is shown as **allow-update (none;);**. Since this is BIND 9's default mode it is not required and has been omitted.

An example master.localhost.

reverse-map

Reverse mapping describes the process of translating an IP address to a host name. This process uses a special domain called IN-ADDR.ARPA and, if it is to be supported, requires a corresponding zone file. Reverse Mapping and the required zone files are described in detail.

0.0.127.IN-ADDR.ARPA

This special zone allows reverse mapping of the loopback address 127.0.0.1 to satisfy applications which do reverse or double lookups. Any request for the address 127.0.0.1 using this name server will return the name **localhost**. On its face this may seem a little strange and you can either continue to treat the process as magic or get some understanding of how resolvers work and the unpleasant issue of reverse mapping. The 0.0.127.IN-ADDR.ARPA zone is defined as shown below

```
zone "0.0.127.in-addr.arpa" in{
  type master;
  file "localhost.rev";
};
```

In many examples and even the files supplied with BIND 9 a zone specific option allow-update statement is shown as **allow-update (none;);**. Since this is BIND 9's default mode it is not required and has been omitted.

An example localhost.rev.

Problems, comments, suggestions, corrections (including broken links) or some thing to add? Please take the time from a busy life to 'mail us' (at top of screen), the webmaster (below) or info-support at zytrax. You will have a warm inner glow for the rest of the day.

Chapter 8. DNS Resource Records

DNS records have a binary or wire-format which is used in queries and responses and a text format which is used in a zone files and is described in this chapter.

Contents

DNS Record Types

A full list of DNS Record Types may be obtained from IANA DNS Parameters.

RR	Value	RFC	Description
A	1	RFC 1035	IPv4 Address record. An IPv4 address for a host.
AAAA	28	RFC 3596	IPv6 Address record. An IPv6 address for a host. Current IETF recommendation for IPv6 forward-mapped zones.
A6	38	RFC 2874	Experimental. Forward mapping of IPv6 addresses. An IP address for a host within the zone.
AFSDB	18	RFC 1183	Location of AFS servers. Experimental - special apps only.
CNAME	5	RFC 1035	Canonical Name. An alias name for a host.
DNAME	39	RFC 2672	Experimental. Delegation of reverse addresses (primarily IPv6).
HINFO	13	RFC 1035	Host Information - optional text data about a host.
ISDN	20	RFC 1183	ISDN address. Experimental = special applications only.
KEY	25	RFC 2535	DNSSEC. Public key associated with a DNS name.
LOC	29	RFC 1876	Stores GPS data. Experimental - widely used.
MX	15	RFC 1035	Mail Exchanger. A preference value and the host name for a mail server/exchanger that will service this zone. RFC 974 defines valid names.
NAPTR	2	RFC 3403	Naming Authority Pointer Record. Goss misnomer. General purpose definition of rule set to be used by applications e.g. VoIP

NS	2	RFC 1035	Name Server. Defines the authoritative name server(s) for the domain (defined by the SOA record) or the subdomain.
NSEC	47	RFC 3755	DNSSEC.bis. Next Domain record type. Used with RRSIG.
NXT	30		DNSSEC Next Domain record type. Obsolete use NSEC.
PTR	12	RFC 1035	IP address (IPv4 or IPv6) to host. Used in reverse maps.
RP	17	RFC 1183	Information about responsible person. Experimental - special apps only.
RRSIG	46	RFC 3755	DNSSEC.bis. Signed RRset.
RT	21	RFC 1183	Through-route binding. Experimental - special apps only.
SOA	6	RFC 1035	Start of Authority. Defines the zone name, an e-mail contact and various time and refresh values applicable to the zone.
SRV	33	RFC 2872	Defines services available in zone e.g. ldap, http etc..
SIG	24	RFC 2931//2535	DNSSEC. Signature - contains data authenticated in a secure DNS. RFC 2535.
TXT	16	RFC 1035	Text information associated with a name. The SPF record is defined using a TXT record but is not (July 2004) an IETF RFC.
WKS	11	RFC 1035	Well Known Services. Deprecated in favour of SRV.
X25	19	RFC 1183	X.25 address. Experimental - special apps only.

Zone File Directives
- $ORIGIN
- $INCLUDE
- $TTL
- $GENERATE (non-standard BIND only)

Zone File Format

The DNS system defines a number of Resource Records (RRs). The text representation of these records are stored in zone files.

Zone file example

```
; zone file for example.com
```

```
$TTL 2d    ; 172800 secs default TTL for zone
@             IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2003080800 ; se = serial number
                        12h        ; ref = refresh
                        15m        ; ret = update retry
                        3w         ; ex = expiry
                        3h         ; min = minimum
                        )
              IN      NS      ns1.example.com.
              IN      MX  10  mail.example.net.
joe           IN      A       192.168.254.3
www           IN      CNAME   joe
```

The above example shows a very simple but fairly normal zone file. The following notes apply to zone files:

1. Zone files consist of Commnents, Directives and Resource Records
2. Comments start with ';' (semicolon) and are assumed to continue to the end of the line. Comments can occupy a whole line or part of a line as shown in the above example.
3. Directives start with '$' and are standardized - $ORIGIN and $INCLUDE (defined in RFC 1035) and $TTL (defined in RFC 2308). BIND additionally provides the non-standard $GENERATE directive.
4. There are a number of Resource Record types defined in RFC 1035 and augmented by subsequent RFCs. Resouce Records have the generic format:

```
5.name ttl  class   rr     parameter
```

   The value of 'parameter' is defined by the record and is described for each Resource Record type in the sections below.

6. The $TTL should be present and appear before the first Resource Record (BIND 9).
7. The first Resource Record must be the SOA record.

DNS Generic Record Format

Resource Records have two representations. A textual format described in this chapter and a binary or *wire-format* described in Chapter 15.

The textual format has the following generic form:

```
name  ttl  class  type  type-specific-data
```

Where:

name	The name of the node in the zone file to which this record belongs
ttl	32 bit value. The time to Live in seconds (range is 1 to x). The value zero indicates the data should not be cached.
class	A 16 bit value which defines the protocol family or an instance of the protocol. The normal value is IN = Internet protocol (other values are HS and CH both historic MIT protocols).
types	The resource record type which determines the value(s) of the *type-specific-data* field. Type takes one of the values below.
type-specific-data	Data content of each record is defined by the *type* and *class* values.

The generic binary or wire-format is:

```
name  ttl class type rdlen rdata
```

The binary format is described in chapter 15 RR format

DNS Zone File Directives

Directives start with '$' and are standardized - $ORIGIN and $INCLUDE (defined in RFC 1305) and $TTL (defined in RFC 2308). BIND additionally provides the non-standard $GENERATE directive.

Directive	Description
$INCLUDE	Includes the defined file in-line.
$ORIGIN	Defines the base name (aka label) to be used for 'unqualified' name substitution.
$TTL	Defines the default Resource Record TTL value, used if no TTL is defined in a resource record.

Chapter 9. HOWTOs

This chapter defines HOWTOs for a number of commonly requested features.

HOWTO - Configure Load Balancing

This HOWTO assumes you want the DNS server to respond with different addresses in order to provide a simple load balancing solution. You have a choice of solutions based on what you want to do:

Contents

Balancing Mail

Using the MX record you can balance mail in two ways. You can also configure DNS to provide a kinda mail service fail-over.

1. Define multiple MX records with the same priority e.g.

```
; zone file fragment
        IN  MX  10  mail.example.com.
        IN  MX  10  mail1.example.com.
        IN  MX  10  mail2.example.com.
....
mail  IN  A       192.168.0.4
mail1 IN  A       192.168.0.5
mail2 IN  A       192.168.0.6
```

The name server will deliver the MX records in the order defined by the rrset-order and the receiving SMTP software will select one based on its algorithm. In some cases the SMTP alogithm may work against the definition of the rrset-order statement. Current versions of sendmail (8.13.x), Exim (4.44) and Postfix (2.1 or 2.2) all have definitive references to indicate they randomly select equal preference servers (Postfix allows control of the behaviour with the smtp_randomize_addresses parameter) and consequentially may use an address which the *rrset-order* has carefully tried to change! qmail, courier-mta and Microsoft (Exchange and IIS SMTP) documentation do not appear to have definitive references to indicate how they handle this case.

10. The alternate approach is to define multiple A records with the same name and different IP addresses.

```
; zone file fragment
         IN  MX  10  mail.example.com.
....
mail           IN  A       192.168.0.4
         IN  A       192.168.0.5
         IN  A       192.168.0.6
```

In this case the load-balancing effect is under the control of BIND and the rrset-order record. In order to avoid problems if the receiving mail system does reverse look-up as a spam check then the PTR records for 192.168.0.4, 192.168.0.5, 192.168.0.6 above must all define to mail.example.com.

In all the above cases each mail server must be capable of handling and synchronising the load for all the mail boxes served by the domain, using some appropriate back-end to do this or by defining all but one server to be a relay or forwarder.

Balancing Other Services

Assuming you want to load share your ftp or web services then you simply define multiple A records with the same name and different IPs as in the example below.

```
; zone file fragment

ftp   IN  A   192.168.0.4
ftp   IN  A   192.168.0.5
ftp   IN  A   192.168.0.6
www   IN  A   192.168.0.7
www   IN  A   192.168.0.8

; or use this format which gives exactly the same result
ftp   IN  A   192.168.0.4
      IN  A   192.168.0.5
      IN  A   192.168.0.6
```

```
www    IN  A    192.168.0.7
       IN  A    192.168.0.8
```

The DNS will deliver all the IP addresses defined, the first IP address in the list will be in a default round robin (controlled by the rrset 'named.conf' directive). The FTP and WEB servers must all be exact replicas of each other in this scenario.

Balancing Services

The SRV record provides the kind of fine control that you are probably looking for to balance load with a fine level of granularity as well as provide some level of fail-over. It provides both *priority* and *weight* fields for the purpose. The SRV record description contains an example illustrating this kind of flexibility.

Controlling the order of RRs

You can control the order of RR that BIND supplies in response to queries by use of a rrset-order **option** which works for any set of equal records. The default behaviour is defined to be random-cyclic - a random selection of the initial order thereafter cyclic (round-robin). Experimentation with BIND 9.3.0 showed that the default is cyclic.

Effectiveness of DNS Load Balancing

Assuming the interest in controlling the order is to load balance across multiple servers supporting a single service - the real question is how effective can the DNS system be in providing this balancing?

The effects of caching will distort the effectiveness of any IP address allocation algorithm unless a 0 TTL is used which has the effect of significantly increasing the load on the DNS (and is not always implemented consistently). In this case the cure may be worse than the disease **Good news** we have good load balancing on our web servers. **Bad news** we need 17 more DNS servers!. Intuitively, and without running any experiments to verify, we would suggest that given a normal TTL (12 hours or more) and ANY IP allocation algorithm other than a single static list, loads should be reasonably balanced (measured by request arrivals at destination IPs) given the following assumptions:

1. traffic is balanced over a number of DNS caches i.e. traffic originates from a number of ISPs or customer locations. Specifically there are no PATHOLOGICAL patterns where 90% (or some large'ish number) of the load originates from a particular cache/service).

2. the volume of traffic is reasonably high - since PATHOLOGICAL patterns are more likely in small traffic volumes.

What DNS load balancing cannot do is to account for service loading e.g. certain transactions may generate very high CPU or resource loads. For this type of control only a local load balancer - one which measures response times - will be effective.

Finally on this topic if you still consider that a DNS solution will do the trick if only you could control the order of IP address generation you can use the BIND 9 SDB API to achieve the result (or one of the available libraries).

HOWTO support http://example.com

This HOWTO configures a DNS server to allow URL's of the form http://www.example.com and http://example.com - both URL's will get to the same web server. Seems its the cool thing to do these days.

Beware: You will also have to change your web server for this to work (change defined below for Apache using Virtual hosts).

```
; zone fragment for 'zone name' example.com
....
; SOA NS MX and other stuff

; define an IP that will resolve example.com
                 IN      A      192.168.0.3
; you could also write the above line as
; example.com.  IN      A      192.168.0.3
www              IN      CNAME  example.com. ; dot essential
; aliases www.example.com to example.com
; OR define another A record for www using same host
; this is the least number of changes and saves a CNAME
www              IN      A      192.168.0.3
```

You could do the above for any other host name e.g. ftp as long as different ports are in use e.g. ftp://example.com would work if your FTP server was appropriately configured and on the same host!

Apache change

Assuming you are using virtual hosts on an Apache server you will have a definition in your httpd.conf file something like this:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName www.example.com
    ErrorLog logs/error_log
    CustomLog logs/access_log common
```

```
</VirtualHost>
```

you need add a second definition with **ServerName** modified to reflect your change as follows:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName example.com
    ErrorLog logs/error_log
    CustomLog logs/access_log common
</VirtualHost>
```

An alternate method is to use a **single** <VirtualHost> with the **ServerAlias** directive as shown below:

```
<VirtualHost 10.10.0.23>
    ServerAdmin webmaster@example.com
    DocumentRoot /path/to/web/root
    ServerName www.example.com
    ServerAlias example.com
    ErrorLog logs/error_log
    CustomLog logs/access_log common
</VirtualHost>
```

Notes:

1. In many cases when you type **example.com** in your browser, the ever helpful browser will auto-complete (or guess) that what you really meant was **www.example.com** and add the **www**. So after all that hard work in many browsers example.com would have worked even if you had done nothing!
2. If you are using MS Frontpage extensions with a single <VirtualHost> definition then the ServerName must be the name that is used to login to FP. In the example above the FrontPage login name used would be www.example.com. When using FP if the ServerName were example.com and the ServerAlias were www.example.com then the FP login would fail.

HOWTO - Configure Sub-domains (a.k.a subzones)

This HOWTO is an overview of sub-domain configuration, where a sub-domain is defined as being:

- zone (domain) name = example.com
- domain host name = bill.example.com
- sub-domain name = us.example.com
- sub-domain host name = ftp.us.example.com

You have a choice of two strategies for handing sub-domain addressing:

1. Fully delegate the sub-domain - in this case you will need one or more name servers for the sub-domain.
2. Create a **virtual** (or pseudo) sub-domain - in this case you define the sub-domain's configuration, as well as the main zone configuration, in a single name-server and zone file.

In this HOWTO we configure a **virtual** sub-domain i.e. the subdomain definition is included in a single zone file.

Zone Name Server Configuration

The primary name server for our domain is running BIND and has a named.conf file that defines the zone.

We received some mail which suggested that we show the explicit use of the allow-transfer statement. The samples in Chapter 6 all show this statement in use but for anyone just using this section it is not apparent.

Zone Name-Server named.conf

The **named.conf** file will contain statements similar to the following fragment defining the main zone as normal:

```
// named.conf file fragment
....
options {
    ....
    // stop everyone
    allow-transfer {"none";};
    ....
};
zone "example.com" in{
   type master;
   file "master/master.example.com";
   // explicitly allow slave
   allow-transfer {192.168.0.4;};
};
```

Zone Name-Server Zone Files

The file 'master.example.com' (or whatever naming convention you use) will contain our domain and sub-domain configuration with, say, a couple of name servers.

```
; zone fragment for 'zone name' example.com
; name servers in the same zone
$TTL 2d ; zone default TT = 2 days
$ORIGIN example.com
@              IN      SOA   ns1.example.com. hostmaster.example.com. (
               2003080800 ; serial number
```

```
                2h          ; refresh =  2 hours
                15M         ; update retry = 15 minutes
                3W12h       ; expiry = 3 weeks + 12 hours
                2h20M       ; minimum = 2 hours + 20 minutes
                )
; main domain name servers
                IN      NS      ns1.example.com.
                IN      NS      ns2.example.com.
; mail servers for main domain
                IN      MX 10   mail.example.com.
; A records for name servers above
ns1             IN      A       192.168.0.3
ns2             IN      A       192.168.0.4
; A record for mail servers above
mail            IN      A       192.168.0.5
; other domain level hosts and services
bill            IN      A       192.168.0.6
....
; sub-domain definitions
$ORIGIN us.example.com.
                IN      MX 10   mail
; record above could have been written as
; us.example.com.    IN   MX 10 mail.us.example.com.
; A record for subdomain mail server
mail            IN      A       10.10.0.28
; the record above could have been written as
; mail.us.example.com. A 10.10.0.28 if it's less confusing
ftp             IN      A       10.10.0.29
; the record above could have been written as
; ftp.us.example.com. A 10.10.0.29 if it's less confusing
....
; other subdomain definitions as required
```

Additional sub-domains could be defined in the same file using the same strategy. For administrative convenience you could use $INCLUDE directives e.g.

```
; snippet from file above showing use of $INCLUDE
....
; other domain level hosts and services
bill            IN      A       192.168.0.5
....
; sub-domain definitions
$INCLUDE us-subdomain.sub
; other subdomain definitions as required
```

HOWTO - Delegate a Sub-domain (a.k.a. subzone)

This HOWTO configures BIND to **fully** delegate the responsibility for a sub-domain to another name server. This is not the only possible method of defining sub-domains

(virtual - or pseudo - subdomains). The following defines the hierarchy we want to create:

- zone (domain) name = example.com
- domain host name = bill.example.com
- sub-domain name = us.example.com
- sub-domain host name = ftp.us.example.com

To ease the administration load we want to fully delegate the responsibility for the administration of the *us* sub-domain (and its reverse-lookup) to the the *us.example.com* management group.

This HOWTO assumes that the name servers for our zone (example.com) are all in our domain. If they are not, the actual configuration is exactly the same but you will have to convince the name server administrator to carry out the configuration. If we own the name servers we can do what we like!

Finally it is important to remember that as far as the internet registration authorities and root name-servers are concerned sub-domains do not exist. All queries for anything which ends with *example.com* will be directed to the name-servers for the *example.com* zone. These name servers are responsible for redirecting the query to the sub-domain name-servers.

For want of any better terminology we call our servers the **domain name-server** (this one is visible to registration authorities) and the the *sub-domain name-server* (essentially visible only to the *domain name-server*).

We received some mail which suggested that we show the explicit use of the allow-transfer statement. The samples in Chapter 6 all show this statement in use but for anyone just using this section it is not apparent.

domain Name Server Configuration

The name servers for our domain are running BIND and has a named.conf file that defines the zone.

Domain Name-Server named.conf

The 'named.conf' file will contain statements similar to the following fragment defining the main zone:

```
// named.conf file fragment
....
options {
    ....
    allow-transfer {"none";};
    ....
};
zone "example.com" in{
          type master;
          file "master/master.example.com";
  // explicitly allow slave
```

```
        allow-transfer {192.168.0.4;};
};
// optional - we act as the slave (secondary) for the delegated domain
zone "us.example.com" IN {
        type slave;
        file "slave/slave.us.example.com";
        masters {10.10.0.24;};
};
```

The optional definition of a slave (secondary) name server for our delegated *us.example.com* sub-domain is probably good practice but not essential - you can define it to be any name server.

Domain Name-Server Zone Files

The file 'master.example.com' (or whatever naming convention you use) will contain our domain configuration with two name servers.

```
; zone fragment for example.com
; name servers in the same zone
$TTL 2d ; default TTL is 2 days
$ORIGIN example.com.
@               IN      SOA   ns1.example.com. hostmaster.example.com. (
                2003080800 ; serial number
                2h         ; refresh =  2 hours
                15M        ; update retry = 15 minutes
                3W12h      ; expiry = 3 weeks + 12 hours
                2h20M      ; minimum = 2 hours + 20 minutes
                )
; main domain name servers
                IN      NS     ns1.example.com.
                IN      NS     ns2.example.com.
; main domain mail servers
                IN      MX      mail.example.com.
; A records for name servers above
ns1             IN      A      192.168.0.3
ns2             IN      A      192.168.0.4
; A record for mail server above
mail            IN      A      192.168.0.5
....

; sub-domain definitions
$ORIGIN us.example.com.
; we define two name servers for the sub-domain
@               IN      NS     ns3.us.example.com.
; the record above could have been written without the $ORIGIN as
; us.example.com. IN NS ns3.us.example.com.
; OR as simply
;      IN NS   ns3
; the next name server points to ns1 above
                IN      NS     ns1.example.com.
; sub-domain address records for name server only - glue record
ns3             IN      A      10.10.0.24 ; 'glue' record
; the record above could have been written as
; ns3.us.example.com. A 10.10.0.24 if it's less confusing
```

Notes:

1. The above fragment makes the assumptions that *ns1.example.com* will act as a slave (secondary) for the *us.example.com* sub-domain. If not we could have defined other name-servers in the same way.

2. The A record for *ns3.us.example.com* for the sub-domain is the so-called *glue* record and MUST be present. It is necessary to allow a DNS query to succeed in a single transaction - which always requires an IP address - which is always defined in an A or AAAA RR.

   Note: All name server queries require both a name and an IP address (a glue record) in the response (answer). In the case of the gTLD or ccTLD servers they provide the glue (IP address) record. These glue records were captured when the domain was registered. The A record for the name server *ns2.example.com* is not strictly speaking a glue record but the A record for *ns1.example.com* IS a glue record for *us.example.com* but NOT, strictly speaking, for *example.com*.

Sub-domain Configuration

Assuming our sub-domain name-server is also running BIND we will have the following configuration.

Sub-domain named.conf

The 'named.conf' file will contain statements similar to the following fragment defining the sub-domain zone:

```
// named.conf file fragment
....
options {
    ....
    allow-transfer {"none";};
    ....
};
zone "us.example.com" in{
          type master;
          file "master/master.us.example.com";
  // explicitly allow slave
          allow-transfer {192.168.0.3;};
};
```

Sub-domain Zone Files

The file *master.us.example.com* (or whatever convention you use) will contain our sub-domain configuration with, say, a couple of name servers.

```
; zone fragment for sub-domain us.example.com
; name servers in the same zone
$TTL 2d ; default TTL = 2 days
$ORIGIN us.example.com.
@              IN     SOA   ns3.us.example.com. hostmaster.us.example.com. (
               2003080800 ; serial number
               2h         ; refresh =  2 hours
               15M        ; update retry = 15 minutes
               3W12h      ; expiry = 3 weeks + 12 hours
               2h20M      ; minimum = 2 hours + 20 minutes
               )
; sub-domain name servers
```

```
                  IN      NS     ns3.us.example.com.
                  IN      NS     ns1.example.com. ; see notes below
; sub-domain mail server
                  IN      MX 10  mail.us.example.com.
; above record could have been written as
;                 IN      MX 10  mail
; A records for name servers above
ns3               IN      A      10.10.0.24
ns1.example.com.  IN      A      192.168.0.3 ; 'glue' record
; A record for mail server above
mail              IN      A      10.10.0.25
; next record defines our ftp server
ftp               IN      A      10.10.0.28
; the record above could have been written as
; ftp.us.example.com. A 10.10.0.24 if it's less confusing
....
; other sub-domain records
....
```

Notes:

1. The above fragment makes the assumptions that our main zone name server will act as a slave (secondary) for *us.example.com*. If not we could have defined other name-servers in the same way.
2. The A record for *ns1.example.com* is a so-called *glue* record and is not strictly necessary since this address will already be available from the initial query to the example.com domain i.e. all queries will descend the domain name hierarchy and already have received the IP for ns1.example.com.
3. Our *ftp* service host (and any others we may define) are only defined in the sub-domain name server and are not visible in the zone name-server.

HOWTO - Configure Mail Servers fail-over

This HOWTO configures a DNS server to provide a 'kinda' fail-over service when the primary mail service is off-line.

Define MX records with different priorities e.g.

```
; zone file fragment
      IN  MX  10 mail.example.com.
      IN  MX  20 mail.example.net.
....
mail  IN  A      192.168.0.4
```

If the most preferred mail server (mail.example.com) is down, mail should be sent to the alternate server (mail.example.net). The server mail.example.net, which in the fragment above is external to the domain and ideally at a separate geographic location, would typically configured as a simple relay (or forwarder) with a very long retry time in which case it will accept the mail and try and relay it to the proper destination (mail.mydomain.com) over the next six weeks or whatever you configure the retry time to be.

Beware: There are a number of articles around the web which suggest that most mail systems do a poor job of using the 'preference system of the MX record.

HOWTO Delegate Reverse Subnet Maps

This HOWTO configures delegated reverse subnet maps as defined in RFC 2317 to support classless routing. There is a longer explanation of reverse mapping which covers the same topic.

Delegated reverse mapping requires the support or participation of your ISP or the Authority that assigned the static IP address range.

Note: When doing a reverse name look-up using your local DNS server before this change would resolve locally without requiring any external DNS access. The configuration defined below will always require to access the zone master for the reverse mapped address IN-ADDR.ARPA domain. If this change has not taken place or has not propagated you will get errors from 'nslookup' or 'dig' operations.

The following fragment shows the 192.168.23.64/27 subnet as a fragment of a reverse map zone file located at the ISP or other Authority that assigned the subnet:

```
$ORIGIN 23.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.isp.com. root.isp.com. (
                              2003080800 ; serial number
                              2h         ; refresh
                              15m        ; update retry
                              2w         ; expiry
                              3h         ; minimum
                              )
              IN  NS      ns1.isp.com.
              IN  NS      ns2.isp.com.
; definition of other IP address 0 - 63
....

; definition of our target 192.168.23.64/27 subnet
; name servers for subnet reverse map
64/27         IN  NS  ns1.mydomain.com.
64/27         IN  NS  ns2.mydomain.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 95 since they are the subnets
; broadcast and multicast addresses not hosts/nodes
65            IN  CNAME   65.64/27.23.168.192.IN_ADDR.ARPA. ;qualified
66            IN  CNAME   66.64/27 ;unqualified name
67            IN  CNAME   67.64/27
....
93            IN  CNAME   93.64/27
94            IN  CNAME   94.64/27
; end of 192.168.23.64/27 subnet
.....
; other subnet definitions
```

The 64/27 construct is an artificial (but legitimate) way of constructing the additional space to allow delegation. This is not technically a domain name and therefore can use '/' (which is not allowed in a domain name) but it could be replaced with say '-' e.g. 64-27 if that makes you more comfortable.

The zone file at the DNS serving the Reverse Map (ns1.mydomain.com in the above example) looks like this:

```
$ORIGIN 64/27.23.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.mydomain.com. root.mydomain.com. (
                                2003080800 ; serial number
                                2h         ; refresh
                                15m        ; update retry
                                2w         ; expiry
                                3h         ; minimum
                                )
              IN  NS      ns1.mydomain.com.
              IN  NS      ns2.mydomain.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 95 since they are the subnets
; broadcast and multicast addresses not hosts/nodes
65            IN  PTR   fred.mydomain.com. ;qualified
66            IN  PTR   joe.mydomain.com.
67            IN  PTR   bill.mydomain.com.
....
93            IN  PTR   web.mydomain.com.
94            IN  PTR   ftp.mydomain.com.
; end of 192.168.23.64/27 subnet
```

Now you have to change your reverse map zone names in the name.conf file to reflect the above change. The following examples shows the reverse map declaration before and after the change to reflect the configuration above:

```
//  before change the reverse map zone declaration would look
// something like this
zone "23.168.192.in-addr.arpa" in{
       type master;
       file "192.168.23.rev";
};
```

Change to reflect the delegated zone name.

```
//  after change the reverse map zone declaration would look
// something like this
zone "64/27.23.168.192.in-addr.arpa" in{
       type master;
       file "192.169.23.rev";
};
```

HOWTO - Define an SPF Record

This section defines HOWTO configure a Sender Policy Framework (SPF) record for a domain and its mail servers.

SPF was initiated by Meng Weng Wong of pobox.com and is being proposed (as of July 2004) as an IETF standard to enable validation of legitimate sources of email. The information below is NOT complete please see the SPF web site which contains further information or the draft RFC.

Briefly the design intent of the SPF record is to allow a receiving MTA (Message Transfer Agent) to interrogate the Name Server of the domain which appears in the email (the **sender**) and determine if the originating IP of the mail (the **source**) is authorized to send mail for the **sender**'s domain.

The SPF information is contained in a standard TXT RR (though a new RR type may be allocated if and when SPF reaches standardization by the IETF).

If a SPF (TXT) RR exists and authorizes the **source** IP address the mail can be accepted by the MTA. If the SPF (TXT) RR does not authorize the IP address the mail can be bounced - it did not originate from an authorized **source** for the **sender**'s domain. If the domain does not have an SPF RR the situation is no worse than before.

Many Open Source MTAs have already been modified to use the SPF record and there is no down-side and plenty of potential up-side to implement the proposed record format now.

We use the following terminology to try and simplify the descriptions below:

1. **sender** - the full email address of the originator of the mail item (typically uses **return-path** in the actual SPF checks)
2. **source-ip** - the IP address of the SMTP server trying to send this message
3. **sender-domain** the domain name part of the **sender**'s email address e.g. assume the **sender** is info@example.com the **sender-domain** is example.com.

The SPF record defines one or more tests to carry out to verify the **sender**. Each test returns a condition code (**pre** below). The first test to **pass** will terminate SPF processing.

TXT RR Format

The standard TXT record format is defined as:

```
name  ttl  class   rr     text
```

The SPF record is entirely contained in the **text** field (a quoted string). SPF defines the contents of the quoted string as follows:

```
v=spf1 [[pre] type ] ... [mod]
```

Where:

Parameter	Description
v=spf1	Mandatory. Defines the version being used. Currently the only version supported is **spf1** .
pre	Optional (defaults to +). **pre** defines the code to return when a match occurs. If a test is conclusive either add + or omit (defaults to +). If a test might not be conclusive use "?" or "~" (tilde). "-"(minus) is typically only used with **-all** to indicate that if we have had no previous matches - fail.

Value	Description
+	Default. Pass.
-	Fail.
~	Softfail.
?	Neutral.

type

Defines the mechanism **type** to use for verification of the sender. May take one of the following values:

Basic Mechanisms

These **types** do NOT define a verification mechanism but affect the verification sequence.

1. **include** - Recurse (restart) testing using supplied domain. The **sender-domain** is replaced with the included domain name. Example:

```
2.    ; spf record for example.com
3.    example.com.  IN TXT "v=spf1 include:example.net -all"
4.    ; use the SPF details for example.net
5.    ; in the above case to replace example.com's SPF
6.    ; or
7.    example.com.  IN TXT "v=spf1 mx include:example.net -all"
8.    ; additive - use MX RR for example.com
9.    ; AND if that fails use example.nets's SPF
10.
11.  all - The all type terminates processing (but may be
    optionally followed by a mod value). It is defined to
    be optional but it is a Good Thing™ to include it. It
    is normally present in the form -all to signify that
    if processing reaches this point without a prior match
    the result will be fail. But if you are not sure that
    the tests are conclusive you could use ?all which
    would allow mail to be accepted even if all previous
    checks failed.
12.
```

Sender Mechanisms

These **types** define a verification mechanism.

1. ip4 - use IP Version 4 addresses e.g. 192.168.3.0 for verification
2. ip6 - use IP Version 6 addresses for verification
3. a - use DNS A RRs for verification
4. mx - use DNS MX RRs for verification
5. ptr - use DNS PTR RRs for verification
6. exists - test for existence of domain

Value	Description

a
a:domain
a:domain/cidr
a/cidr
	In its base form this uses the **sender-domain** to find an A RR(s) to verify the **source**. This form relies on an A RR for the domain e.g.
	
	`; fragment for example.com`
	`$ORIGIN example.com.`
	`example.com.  IN TXT "v=spf1 a -all"`
	`; needs domain A record`
	`@  IN  A 192.168.0.3`
	`; functionally the same as`
	`example.com. IN  A 192.168.0.3`
	
	The form **a/cidr** applies the test to the **cidr** (or IP refix or slash) range of the **sender-domain**'s A RR.
	
	The form **a:domain** replaces **sender-domain** with **domain**'s A RR for verification. This does NOT use **domain**'s SPF record(s) (use include for that). The **domain** form may use macro-expansion features. Example:
	
	`; fragment for example.net`
	`$ORIGIN example.net.`
	`@  IN TXT "v=spf1 a:example.com -all"`
	`; will use a single A query to example.com`
	`; which may not yield the result expected unless`
	`; example.com has an A record as below`
	`@  IN  A 192.168.0.3`
	`; functionally the same as`
	`example.com. IN  A 192.168.0.3`
	
	can take a host name format as shown below:
	
	`; fragment for example.net`
	`$ORIGIN example.net.`
	`@  IN TXT "v=spf1 a:mail.example.com -all"`
	`; will use a single A query for mail.example.com`
	
	The form **a:domain/cidr** applies the cidr range to the IP address obtained from the A query e.g.
	
	`; fragment for example.net`
	`$ORIGIN example.net.`
	`@  IN TXT "v=spf1 a:mail.example.com/27 -all"`
	`; will use a single A query for mail.example.com`

	Any of the 32 IP addresses that contain mail.example.com will pass. e.g. if the **source-ip** is 192.168.0.25 and the A RR for mail.example.net is 192.168.0.2 then the test will pass.

mx
mx:domain
mx:domain/cidr
mx/cidr
	This basic form without any extensions uses the MX RR of the **sender-domain** to verify the mail **source-ip**. The MX record(s) return a host name from which the A record(s) can be obtained and compared with the **source-ip**. The form **mx/cidr** applies the IP Prefix or slash range to the A RR address. With any of the domain extensions the MX record of the designated (substituted) domain is used for verification. The **domain** form may use macro-expansion features.
	
	Warning Remember the **MX** RR defines the receiving MTA. If this is not the same host(s) as the sending (SMTP) MTA tests based on an **mx** type will fail. Examples:
	
	`; fragment for example.com`
	`$ORIGIN example.com.`
	`  IN  TXT "v=spf1 mx:example.net -all"`
	`; verify sender using exmple.net MX and A RRs`
	`; fragment for example.com`
	`$ORIGIN example.com.`
	`  IN  TXT "v=spf1 mx:/26 -all"`
	`; verify sender using exmple.com MX and A RRs`
	`; and use 16 address range`

ptr
ptr/domain
	Use the **source-ip**'s PTR RR and a reverse map query. The AA RR for the host is then obtained. If this IP matches the **sender-ip** AND the **sender-domain** is the same as the domain name of the host obtained from the PTR RR then the test passes. The form **ptr:domain** replaces the **sender-domain** with **domain** in the final check for a valid domain name. The **domain** form may use macro-expansion features. The PTR record is the least preferred solution since it places a load on the IN-ADDR.ARPA (IPv4) or IPV6.ARPA reverse-map domains which generally have less capacity than the gTLD and ccTLD domains. Examples:
	
	`; fragment for example.com`
	`$ORIGIN example.com.`
	`@  IN TXT "v=spf1 ptr -all"`
	`; the effect is to allow any host which is reverse mapped`
	`; in the domain to send mail`

ip4:ipv4
ip4:ipv4/cidr
	In its basic form defines an explicit **ipv4** address to verify the mail **source-ip**. If the **source-ip** is the same as **ipv4** the test passes. May optionally take the form **ipv4/cidr** to define a valid IP address range. Since this **type** incurs the least additional load on the DNS the current draft of the proposed RFC recommends this format. Examples:
	`; fragment for example.com`
	`$ORIGIN example.com.`
	`@  IN TXT "v=spf1 ip4:192.168.0.2 -all"`
	`; if source-ip is 192.168.0.2 test passes`
	`; cidr format`
	`@  IN TXT "v=spf1 ip4:192.168.0.2/27 -all"`
	`; if source-ip is in range 192.168.0.1`
	`; to 192.168.0.31 test passes`

ip6:ipv6
ip6:ipv6/cidr
	In its basic form defines an explicit **ipv6** address to verify the mail **source-ip**. If the **source-ip** is the same as **ipv6** the test passes. May optionally take the form **ipv6/cidr** to define a valid IP address range. Since this **type** incurs the least additional load on the DNS the current draft of the proposed RFC recommends this format. Examples:
	`; fragment for example.com`
	`$ORIGIN example.com.`

```
@            IN TXT "v=spf1 ip6:2001:db8::10 -all"
; if source-ip is 2001:db8:0:0:0:0:0:10 test passes
; cidr format
@            IN TXT "v=spf1 ip4:2001:db8::10/120 -all"
; if source-ip is in range 2001:db8:0:0:0:0:0:0
; to 2001:db8:0:0:0:0:0:FF test passes
```

exists:domain — The existence (any valid A RR) of the specified domain allows the test to pass. Domain may use macro-expansion features.

mod

Two optional record modifiers are defined. If present they should follow the last **type** directive i.e. after the **all**. The current values defined are as follows:

Modifier	Description
redirect=domain	Redirects verification to use the SPF records of the defined **domain**. Functionally equivalent to include but can appear on its own (without a terminating **all**) or can placed after the **all** which means "if all the previous test fail try this redirect". Examples:

```
; fragment for example.com
$ORIGIN example.com.
@            IN TXT "v=spf1 ip4:192.168.0.2 -all
redirect=example.net"
; if source-ip is 192.168.0.2 test passes
; if it fails redirect to example.net
; OR single redirect
@            IN TXT "v=spf1 redirect=example.net"
; only use example.net SPF record
```

exp=txt-rr — The **exp** record if present should come last in a SPF record (after the all if present). It defines a DNS name whose TXT record's text may be returned with any failure message. Example:

```
; domain SPF record
        IN  TXT "v=spf1 mx -all exp=getlost.mydomain.com"
; the getlost TXT record
getlost IN  TXT "You are not authorized to send mail for
the domain"
```

The syntax allowed by this record is significantly more complex (see macro-expansion below.

Macro-Expansion

SPF defines a number of macro-expansion features as defined below:

Modifier	Description
%(c)	Only allowed in TXT records referenced by the exp field. The IP of the receiving MTA.
%(d)	The current domain normally the sender-domain %(o) but replaced by the value of any domain argument in the type above.
%(h)	The domain name supplied on HELO or EHLO, normally the hostname of the sending SMTP server.
%(i)	sender-ip The IP of SMTP server sending mail for user info@example.com.
%(l)	replace with local part of sender e.g. if sender is infor@example.com local part is info.

%(o)	The sender-domain e.g. if email address is info@example.com the sender-domain is example.com.
%(p)	The validated domain name. The name obtained using the PTR RR of the sender-ip. Use of this macro will require an additional query unless a ptr type is used.
%(r)	Only allowed in TXT records referenced by the exp field. The name of the host performing the SPF check. Normally the same as the receiving MTA.
%(t)	Only allowed in TXT records referenced by the exp field. Current timestamp.
%(s)	Replace with sender email address e.g. info@example.com
%(v)	Replaced with "in-addr" if sender-ip is an IPv4 address and "ipv6" if an IPv6 address. Used to construct reverse map strings.

The above macros may take one or more additional arguments as follows:

1. r - Indicates reverse the order of the field e,g, %(or) would be displayed as com.example and %(ir) would display 192.168.0.2 as 2.0.168.192. The normal split used "." (dor) as the separator but any other separator may be used e.g. %(sr@) would display example.com.info, when fields are rejojned they will always use a "." (dot).
2. digit - the presence of a digit (range 1 to 128) limits the number of right most elements displayed e.g. %(d1) displays com only from example.com but %(d5) would display up to five right hand element up to the maximum available e.g. example.com.

Examples

Example 1

Example 1: Assumes a single mail server which both sends and receives mail for the domain.

```
; zone file fragment for example.com
$ORIGIN example.com.
              IN  MX 10 mail.example.com.
....
mail          IN  A     192.168.0.4
; SPF stuff
; domain SPF
example.com.  IN  TXT   "v=spf1 mx -all"
; mail host SPF
mail          IN  TXT   "v=spf1 a -all"
```

Notes:

1. the domain SPF is returned from a **sender-domain** query using the **sender**'s email address e.g. **sender** = info@example.com **sender-domain** = example.com. The SPF record only allows the MX host to send for the domain.
2. the mail host SPF is present **in case** the receiving MTA uses a reverse query to obtain the **source-ip** host name and then does a query for the SPF record of that host. The SPF record states that the A record of mail.example.com alone is permitted to send mail for the domain.

If the domain contains multiple MX servers the domain SPF would stay the same but each mail host should have a SPF record.

Example 2

Example 2: Assumes the domain will send mail through an offsite mail server e.g. an ISP:

```
; zone file fragment for example.com
$ORIGIN example.com.
            IN  MX 10  mail.offsite.com.
....
; SPF stuff
; domain SPF
example.com. IN  TXT    "v=spf1 include:offsite.com -all"
; WARNING: offsite.com MUST have a valid SPF definition
```

Notes:

1. This format should be used IF AND ONLY IF you know that **offsite.com** has a valid SPF configuration.
2. **include** recurses (restarts) verification using the SPF records for offsite.com. Mail configuration changes are localised at offsite.com which may simplify administration.
3. **include** could have been replaced with redirect.

Example 3

Example 3: Assumes we are the host for a number of virtual mail domains and that we can send mail from any host in our subnet.

Zone file fragment for one of the virtual mail domains:

```
; zone file fragment for vhost1.com
$ORIGIN example.com.
           IN  MX 10 mail.example.com.
....
; SPF stuff
; domain SPF
vhost1.com. IN  TXT   "v=spf1 include:example.com -all"
```

Notes:

1. the domain SPF is returned from a **sender-domain** query using the **sender**'s email e.g. **sender** = info@vhost1.com, **sender-domain** = vhost1.com. The SPF record recurses to the DOMAIN example.com for verification.

Zone file for example.com

```
; zone file fragment for example.com
             IN  MX 10   mail.example.com.
....
; SPF stuff
```

```
; domain SPF - any host from
; 192.168.0.1 to 192.168.0.30 (32 - bcast and mcast = 30)
; can send mail
example.com. IN  TXT    "v=spf1 ip4:192.168.0.3/27 -all"
; mail SPF
mail         IN  TXT    "v=spf1 ip4:192.168.0.3/27 -all"
```

Notes:

1. the domain SPF is returned from a **sender-domain** query using the **sender**'s email e.g. **sender** = info@example.com **sender-domain** = example.com. The SPF record allows any host in the 32 address subnet which contains 192.168.0.3 to send mail for this and any host virtual domain e.g virtual1.com in the above example. NOTE: while /27 allows 32 IP addresses subnet rules remove 192.168.0.0 and 192.168.0.31 as the multicast and broadcast addresses respectively. [read more about IPv4 Classes]
2. In the above scenario we could have used a slightly shorter version such as:

```
3.    example.com. IN  TXT    "v=spf1 mx/27 -all"
```

   This record has the same effect as **a:192.168.0.3/27** above but will cost a further DNS look up operation whereas the IP is already available.

4. The above scenario relies on the fact that customers will only send mail via the domain example.com i.e. they will NOT send via another ISP at home or when travelling. If you are not sure if this is the case you can terminate the sequence with **?all** which says **kinda pass (soft fail)** and let the mail go through - perhaps logging the incident to capture statistics.

If the domain contains multiple MX servers the domain SPF would stay the same but each mail host should have a SPF record.

Example 4

Example 4: Assumes that the domain never sends mail from ANY location - ever. Typically you would do this to prevent bogus mail for everyone else - it is a supreme act of self-sacrifice!

```
; zone file fragment for example.com
; zone does NOT contain MX record(s)
...
; SPF stuff
; domain SPF
example.com. IN  TXT   "v=spf1 -all"
```

Notes:

1. This SPF test will always fail since the only condition it tests is the **all** which results in a **fail**.

Example 5

Example 4: Uses various macro expansion features:

```
; zone file fragment for example.com
$ORIGIN example.org.
            IN  MX 10 mail.example.com.
....
; SPF records
; domain SPF
@           IN  TXT   "v=spf1 exists:%(ir).%(v).arpa -all ext=badguy.example.com"
badguy      IN  TXT  "The email from %(s) using SMTP server at %(i)
                      was rejected by %(c) (%(r)) at %(t) because it failed
                      the SPF records check for the domain %(p).
                      Please visit http://abuse.example.com/badguys.html
                      for more information"
```

Notes:

1. The badguy TXT above is split across multiple lines for presentation reasons only and should appear on a single line in the zone file.
2. The exists:%(ir).%(v).arpa tests is a great example BUT IT WILL NOT WORK because the **exists** type uses an A RR. But it's a great example to show the power of macro-expansion and we can't think of a better one. Sorry about that.

Chapter 5. Bind on Win2k and NT 4.0

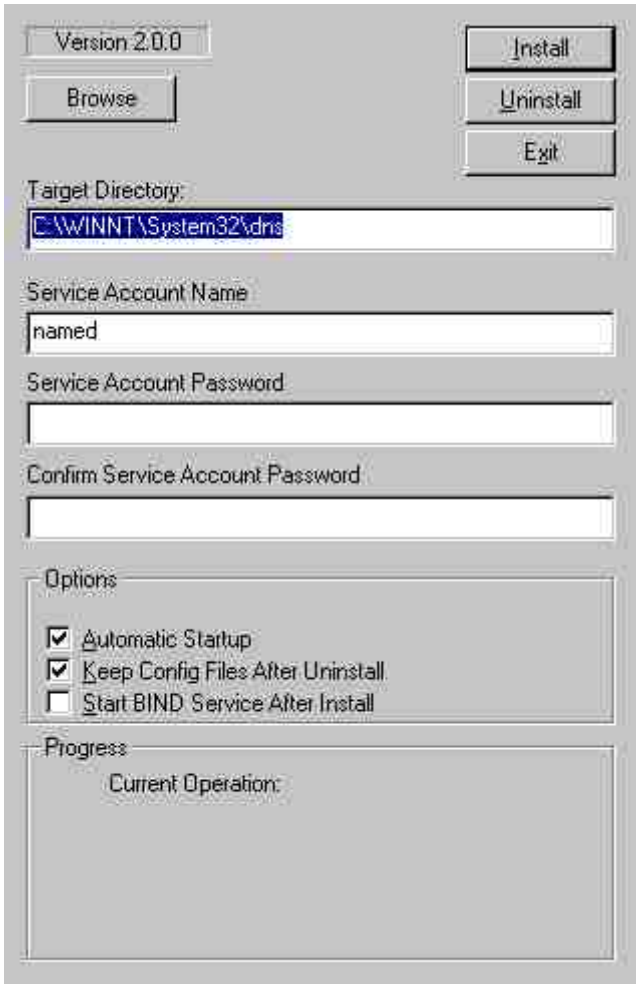
WinNT 4.0 and Win2k Installation

We decided that it was time to try out BIND 9 on our aging NT 4.0 desktops and a Win2000 server that we keep around for some light relief. We primarily wanted to use dig consistently across all our systems so that we could forget *nslookup* and this seemed like the ideal way to do it and provide some local DNS cache services as well on all our remaining NT 4.0 desktops. We had a couple of problems mostly due to Bind's normal paucity of documentation. Still its good to see that some things in life don't change.

Install on NT 4.0

We took a low risk approach to set up a simple caching DNS server and defaulted everything. This was what we did:

1. We down-loaded Bind 9.3.0.zip from the ISC site and unzipped it into a temporary location. So far so good.
2. We found the readme1st.txt file in the temporary install directory and took the 15 seconds required to read this copious document. The most interesting thing at this point is that Bind9 is going to run as an NT service and that it appears it will require a unique NT account, passwords and special permissions. Very unwindows like.
3. From here on out we are logged in as a local administrator on the NT 4.0 PC.
4. We added a new user account called **named** (the default assumed by the BIND install) using Start->programs->administrative tools->user manager. Entered passwords and set *user can't change password* and *password never expires* options. Otherwise it's a normal account so far - but it requires NT service logon capabilities. So we have a little more work to do. We subsequently installed BIND 9 on Windows 2000 and discovered the install process will create the required **named** account automatically so you can bypass this step.
5. In User Manager select *User Rights* from *Policies* menu, check the *Show Advanced User Rights* and then find and select the *log on as a service* right and click the *Add* button.
6. Select the local PC if you are working in a domain and because, by default, it only shows the groups click *Show Users* then select the *named* account and click the *Add* button on this window to assign the *log on as a service* to the *named* account. We did not remove any rights so we suspected the BIND install would object during the install because the readme1st.txt file suggests that if there any more permissions than the **one** required (*log on as a service*) it will send you a little message. If you are allergic to messages and have a couple of minutes to spare you can remove the excess rights. We didn't - we love chatty messages and were quite looking forward to it! Kill all the User Manager windows and we're done with this phase. Again as we subsequently discovered the install process automatically creates the relevant account with the correct permissions.
7. Back to our temporary directory and double click the BindInstall.exe and up pops this screen:



We just added the password for the *named* account and noted in passing that the default install directory is c:\Winnt\system32\dns (or %SystemRoot%\system32\dns in windows terms) and clicked the *Install* button. The install appeared to run like a dream. And yes, we got our little message about too many permissions. It was the highlight of the install. **Note:** We did not check the box labelled *Start BIND Service after Install*, we have some more stuff to do before run the service.

8. We set up a directory called c:\Winnnt\system32\dns\etc\named and then created three sub-directories called run, zones and log. You can do this anywhere but we were a little suspicious of Bind9's ability to figure out Windows paths so we did it this way.

   We placed our master.localhost, localhost.rev and root.servers files in the zones sub-directory and the named.conf file below into the %SystemRoot%\system32\dns\etc directory.

```
// generated by ME
// CACHING NAME SERVER for NT 4.0
// 1. dec 2004
//      a. changed directory statement to windows format
//      b. changed location of log file to named\log\named.log
//      c. changed location of all zone files to named\zones
```

```
//      d. added pid-file directive in named\run\named.pid
options {
        directory "C:\Winnt\system32\dns\etc";
        // version added for security otherwise may be able to exploit known
weaknesses
        version "not currently available";
        pid-file "named\run\named.pid";
        recursion yes;
};

// log to named\log\named.log events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// failure messages up to this point are in the event log
        logging{
        channel my_log{
                  file "named\log\named.log" versions 3 size 250k;
                  severity info;
        };
        category default{
                  my_log;
        };
};
zone "." {
        type hint;
        file "named\zones\root.servers";
};

zone "localhost" in{
        type master;
        file "named\zones\master.localhost";
        allow-update{none;};
};
zone "0.0.127.in-addr.arpa" in{
        type master;
        file "named\zones\localhost.rev";
        allow-update{none;};
};
```

9. We used windows explorer to give the account *named* full control over the directory c:\Winnt\system32\dns with the *inherit* property set so all lower directories pick up the same permissions. The install process does not set permissions - as we subsequently discovered on Windows 2000 - so this step is essential.
10. We are lazy and forgetful so decided to add the BIND9 bin directory (%SystemRoot%\system32\dns\bin) to the Windows path since we are going to use *dig* for sure on a regular basis and in a couple of days we will have forgotten where is it - scrub that idea - in a couple of hours we will have forgotten where it is. So Start->settings->control panel->system and check the environment tab. Find and click the Path and **Add** the following (or wherever your BIND9 bin directory is).
11. `;%SystemRoot%\system32\dns\bin`

    Click *Set* and exit. **Note:** the separator on windows is a semi-colon not a colon as in the *nix world. Cost us 10 minutes with a magnifying glass to fix that one.

12. Time to start the service. Start->settings->control panel->services. Select ISC BIND and try and start it. It failed for us with a login error. So we re-entered the password (using *User Manager*) and tried again and it worked.

The standard install is set for automatic start so we re-booted the PC and checked that *named.exe* was started with *Task Manager*. It was.

13. Finally we opened a dos box and tried a dig command. Seemed to run a bit slowly but we got our results. And yes we had already forgotten where we installed bind. Thank goodness we set the path variable.

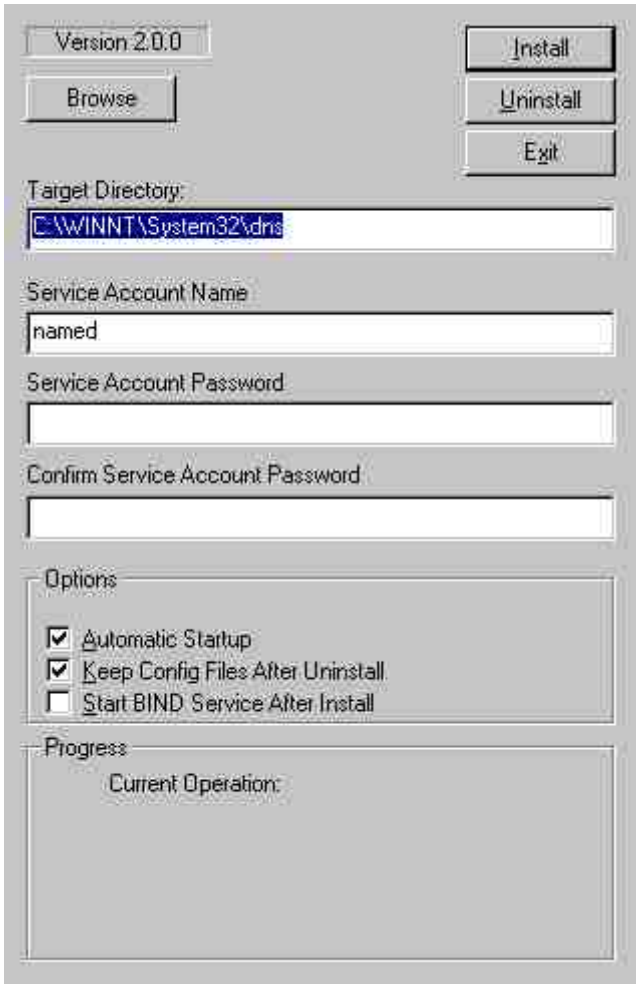
We have not used the service frequently but we were pleasantly surprised at how easy it was to install. Task Manager shows about 380K of memory usage which is by no means excessive. If you want consistently of DNS across a mixed Windows and *nix environment using BIND is the only way to do it.

Win2k Installation

This section describes installation of BIND 9.3.0 on Windows 2000 server.

Having setup BIND on NT 4.0 we decided to install BIND on a server - again as a simple caching server. This was what we did:

1. We down-loaded Bind 9.3.0.zip from the ISC site and unzipped it into a temporary location. So far so good.
2. We found the readme1st.txt file in the temporary install directory and took the 15 seconds required to read this copious document. The most interesting thing at this point is that Bind9 is going to run as an NT service and that it appears it will require a unique NT account, passwords and special permissions. The install process creates the required account but we manually set the account up under NT 4.0 entirely due to a misplaced mistrust in BIND's install process and because the documentation did not tell us it would do so.
3. In our temporary directory we double clicked BindInstall.exe and up pops this screen:



The password entry is optional - you can leave it blank or not as you choose - we left it blank (but see NT 4.0). We noted in passing that the default install directory is c:\Winnt\system32\dns (or %SystemRoot%\system32\dns in windows terms) and clicked the *Install* button. The install appeared to run like a dream. We did not check the box labelled *Start BIND Service after Install*, we have some more stuff to do before run the service.

4. We set up a directory called c:\Winnnt\system32\dns\etc\named and then created three sub-directories called run, zones and log. You can do this anywhere but we were a little suspicious of Bind9's ability to figure out Windows paths so we did it this way.

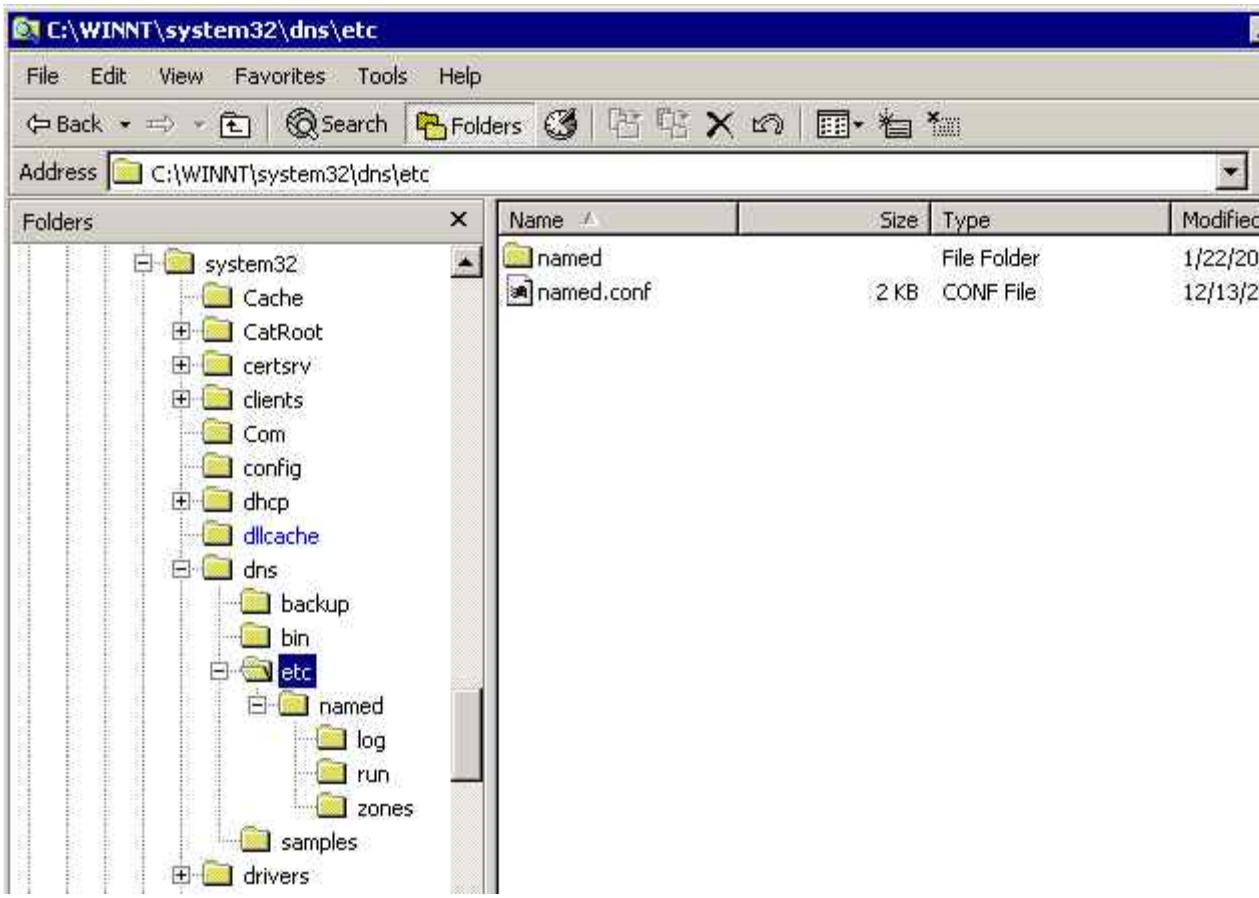
   We placed our master.localhost, localhost.rev and root.servers files in the zones sub-directory and the named.conf file below into the %SystemRoot%\system32\dns\etc directory.

```
// generated by ME
// CACHING NAME SERVER for NT 4.0
// 1. dec 2004
//      a. changed directory statement to windows format
//      b. changed location of log file to named\log\named.log
//      c. changed location of all zone files to named\zones
```

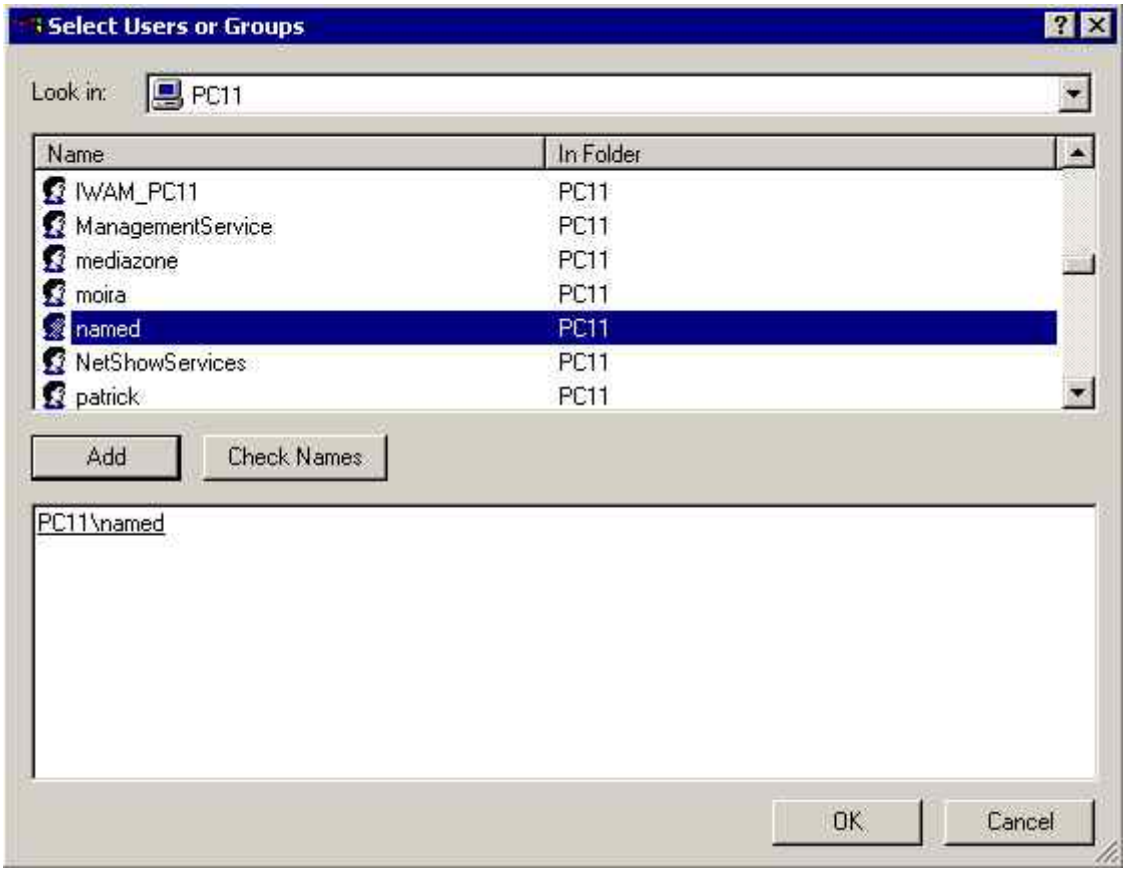
```
//      d. added pid-file directive in named\run\named.pid
options {
        directory "C:\Winnt\system32\dns\etc";
        // version added for security otherwise may be able to exploit known
weaknesses
        version "not currently available";
        pid-file "named\run\named.pid";
        recursion yes;
};

// log to named\log\named.log events from info UP in severity (no debug)
// defaults to use 3 files in rotation
// failure messages up to this point are in the event log
        logging{
        channel my_log{
                  file "named\log\named.log" versions 3 size 250k;
                  severity info;
        };
        category default{
                  my_log;
        };
};
zone "." {
        type hint;
        file "named\zones\root.servers";
};

zone "localhost" in{
        type master;
        file "named\zones\master.localhost";
        allow-update{none;};
};
zone "0.0.127.in-addr.arpa" in{
        type master;
        file "named\zones\localhost.rev";
        allow-update{none;};
};
```

5. We used windows explorer to give the account *named* everything except full control over the directory c:\Winnt\system32\dns with the *inherit* property set so all lower directories pick up the same permissions. This is essential to avoid permission errors when you start the BIND service.

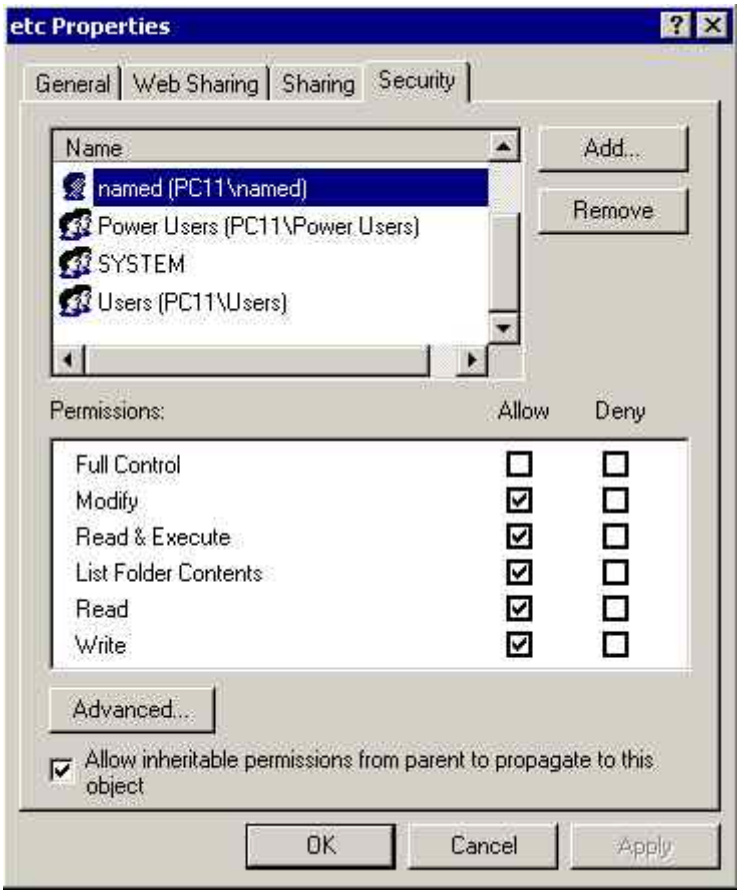
   Select the BIND install directory (this shows the default in c:\%SystemRoot%\system32\dns)in Windows Explorer:



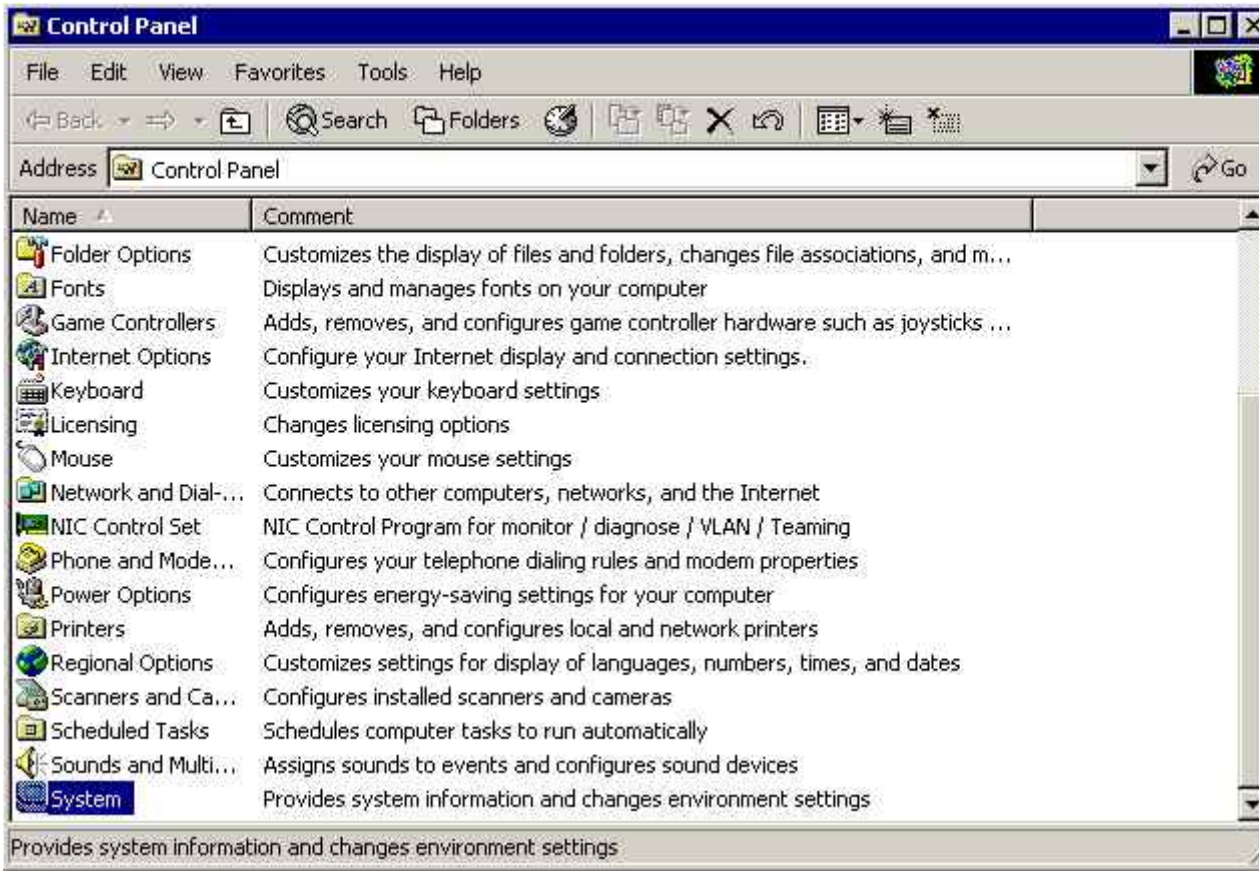
Right click *properties* then *permissions* and find and select the named account and *Add*:



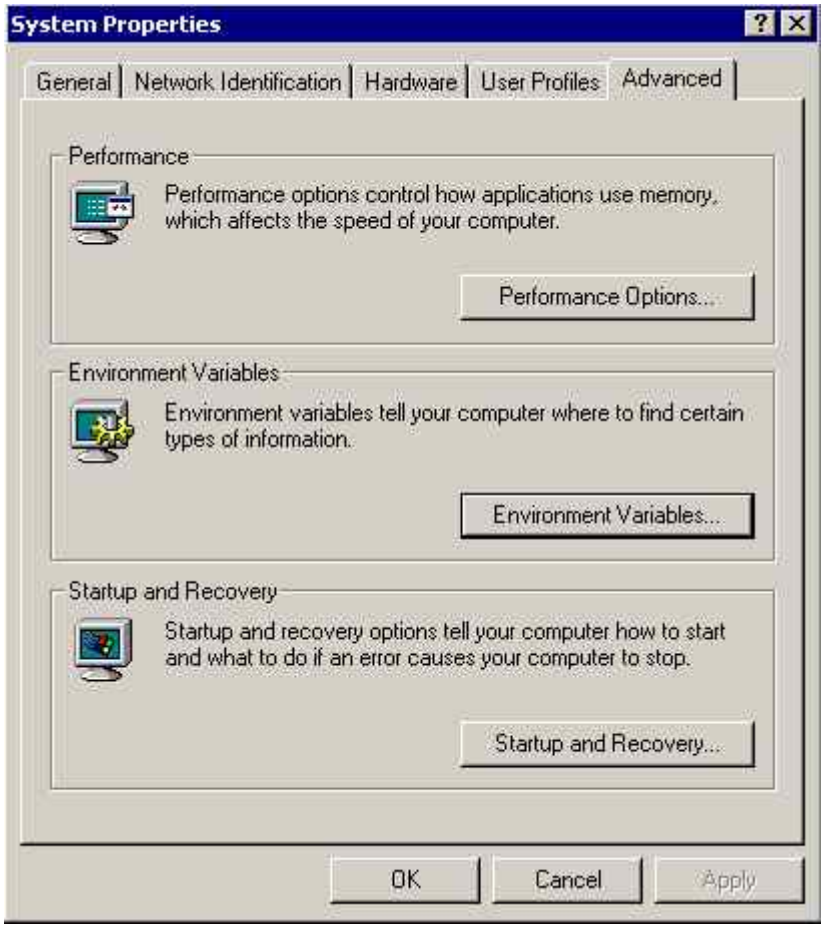
Add all permissions except full control and leave the *inherit* check box set (the default):



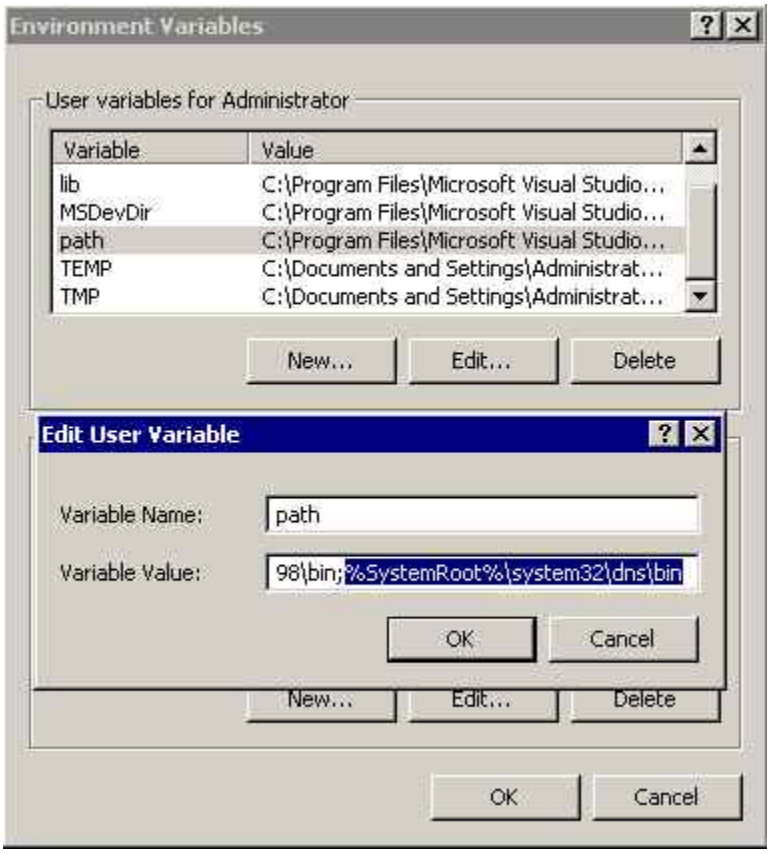
6. We are lazy and forgetful so decided to add the BIND9 bin directory (%SystemRoot%\system32\dns\bin) to the Windows path since we are going to use *dig* for sure on a regular basis and in a couple of days we will have forgotten where is it - scrub that idea - in a couple of hours we will have forgotten where it is. So Start->settings->control panel->system:



Click Environment Variables from the Advanced tab:

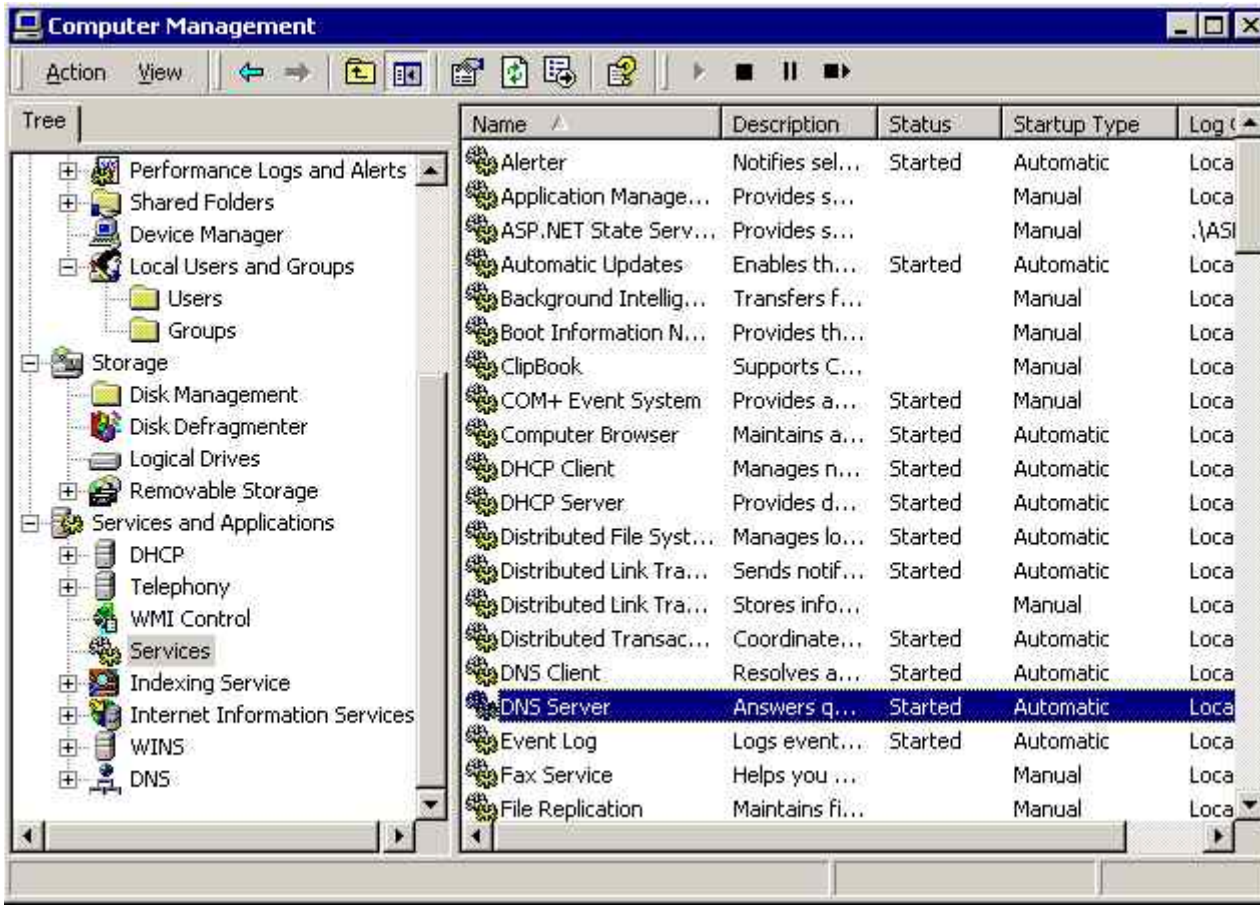


Find and double click the path line and **Add** the following (or wherever your BIND9 bin directory is located):



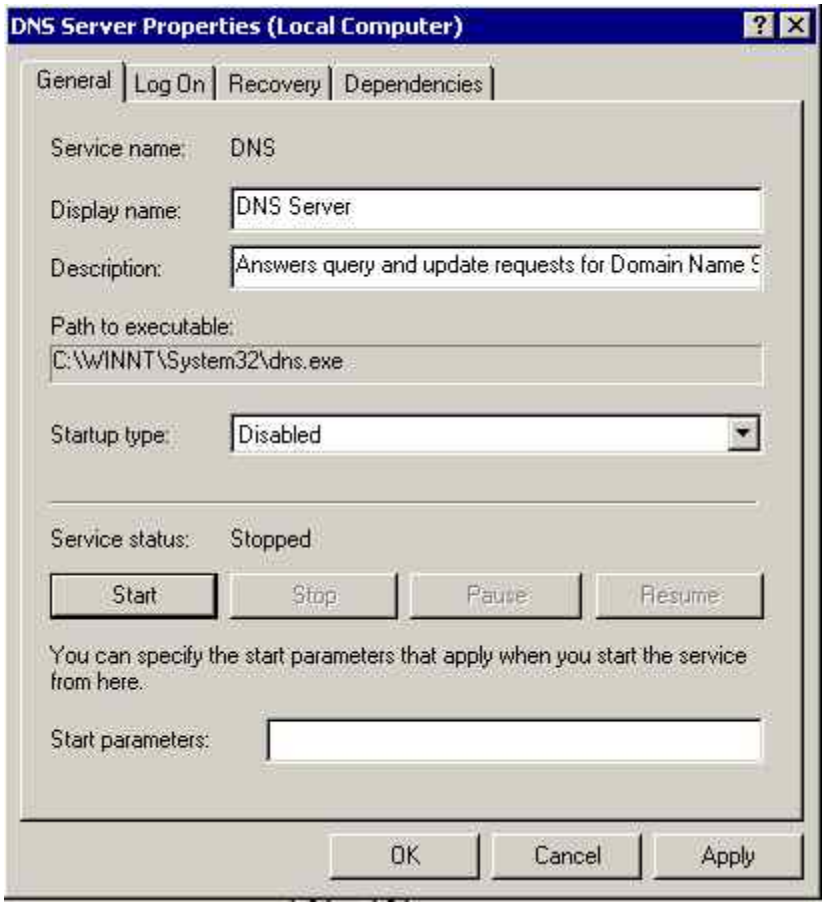
Click *OK* and exit. **Note:** the separator on windows is a semi-colon not a colon as in the *nix world. Cost us 10 minutes with a magnifying glass to fix that one.

7. Time to start the BIND service. This is a Server so the standard Windows DNS services is activated by default. First we have to disable it using Computer Management; expand *Services and Applications* then double click *Services* and find DNS Server:



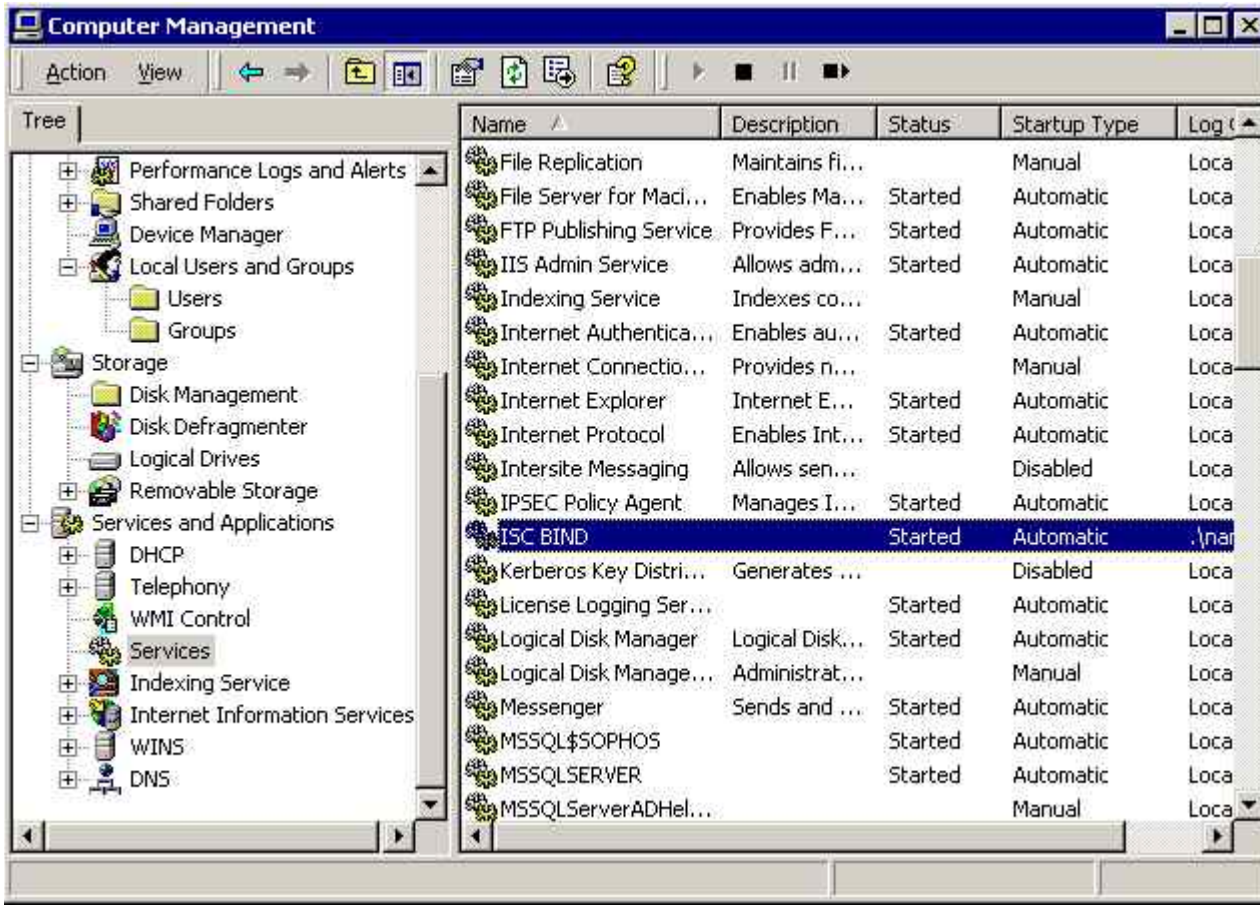
Double click DNS Server and disable the service then click *Stop* then *OK*

:



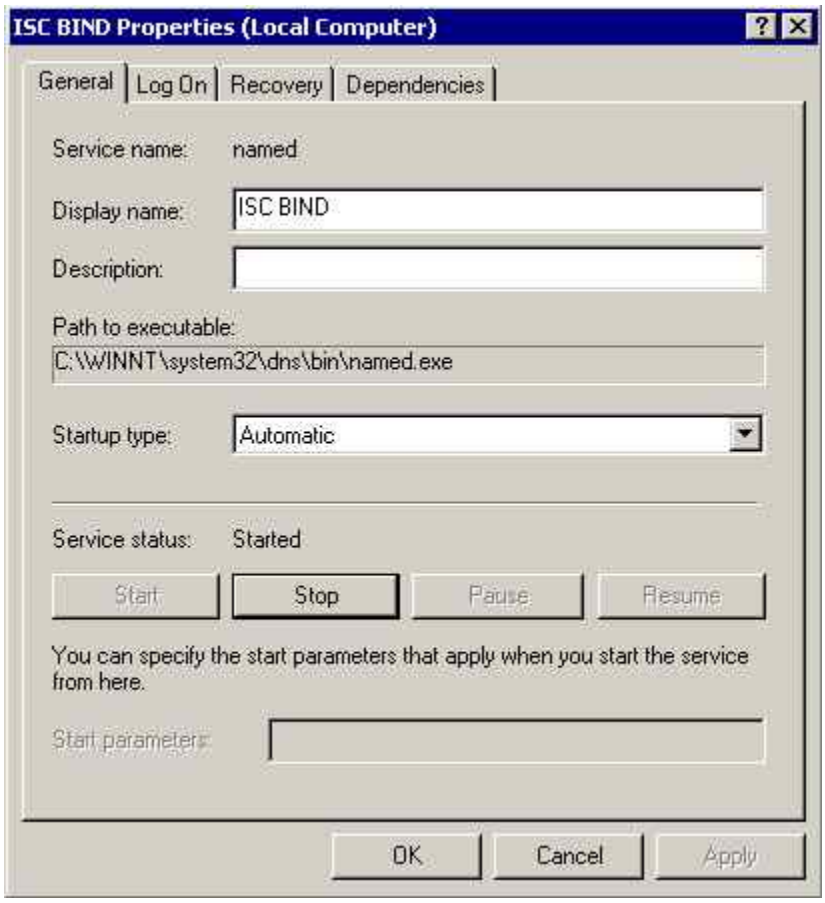
Find and double click ISC BIND

:



Double click ISC BIND and click *Start* (the service is set to *Automatic* by default which means it will load on start-up

:



Any errors will be logged under *Applications* in the *Event Log*.

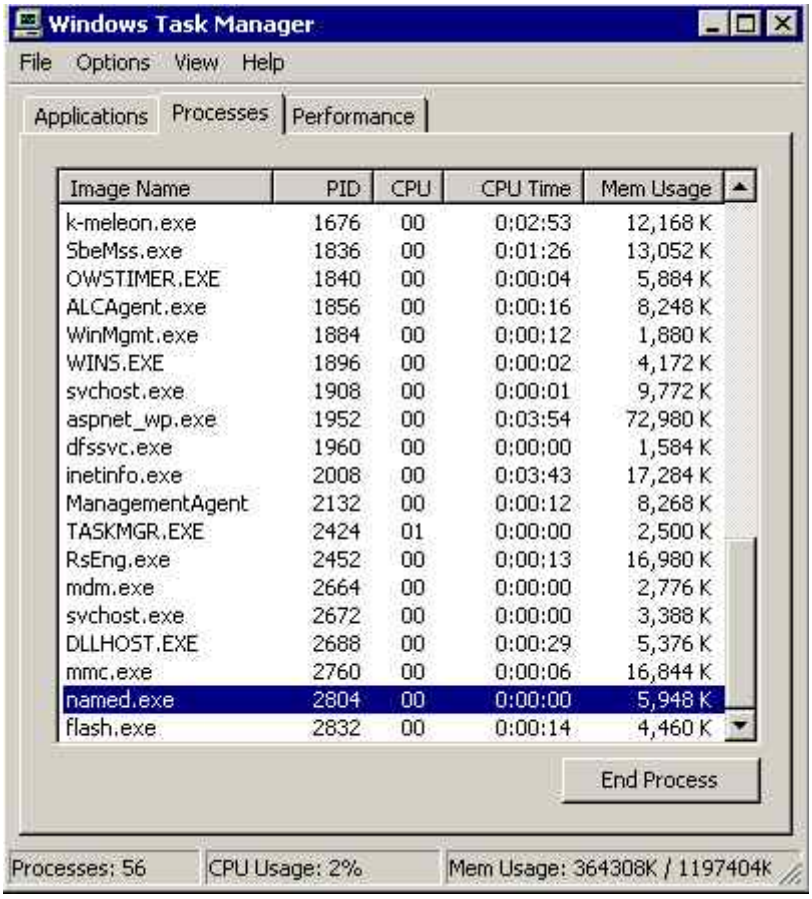
8. Finally we opened a DOS box (Start->run->cmd) and tried a dig command:

```
9. dig @192.168.2.2 example.com any
```

Seemed to run a bit slowly but we got our results. And yes we had already forgotten where we installed bind. Thank goodness we set the path variable.

Note: The @192.168.2.2 is required to force use of the local service irrespective of the TCP configuration.

10. We changed the DNS settings to use the local DNS service (Start->Control Panel->Network and dial-up Connections->select Local LAN->Properties->find and double click Internet TCP/IP) and re-booted the PC. We used *Task Manager* to check that the ISC BIND service started-up (is loads as *named.exe*):



We were pleasantly surprised at how easy it was to install. If you want consistently of DNS for maintenance and other purposes across a mixed Windows and *nix environment using BIND is the only way to do it. As serious side benefit you get *dig* and other tools as a bonus.

Chapter 10. DNS Diagnostics and Tools

This chapter defines tools that may be provided with BIND releases, are generally available or just jolly useful! The tools described either provide specific services or may help in diagnosing problems.

Contents

10.1 Overview and Introduction

10.2 NSLOOKUP

nslookup is officially deprecated in favour of dig. It is however almost universally available - even when **dig** is not - this especially true on windows systems where **dig** is still pretty exotic. You are still more likely to come across **nslookup** than **dig** on a wide range of platforms. Old utilities do not die they just slowly fade away!

Command Format

nslookup in general returns A or PTR records but specific options can be used to override the default. There are both command line and interactive formats available.

nslookup maintains a set of configuration parameters (that may be modified) to add power to the command line. These parameters can be displayed using the **-all** (or set all in interactive mode) argument.

Quick Usage examples

The following are quick examples of common usage - all the options are explained below in mind numbing detail:

```
# lookup a specific host
nslookup www.example.com
# get MX and NS records for the domain
nslookup -type=ANY example.com
# get SOA record and display all nslookup default parameters
nslookup -all -type=SOA example.com
```

The generic command format is:

```
# format 1 lookup target using default DNS server
nslookup [-opt] target
# format 2 lookup target using the specific dns
nslookup [-opt] target dns
# format 3 enter interactive mode using default DNS server
nslookup [-opt]
# format 4 enter interactive mode using the specific dns
nslookup [-opt] - dns
```

Simple examples

Format 1 - Host lookup

```
nslookup www.example.com
# will return
Server: ns1.example.com
Address: 192.168.2.53

Name: www.example.com
```

```
Address: 192.168.2.80
```

Returns the A record for www.example.com using the default DNS server - in this case ns1.example.com (defined in Windows Network Properties or /etc/resolv.conf in *nix systems).

Format 1 - Reverse MAP IP lookup

```
nslookup 192.168.2.80
# will return
Server: ns1.example.com
Address: 192.168.2.53

Name: www.example.com
Address: 192.168.2.80
```

Returns the PTR record for 192.168.2.80 using the IN-ADDR.ARPA domain hierarchy.

Format 2 - Host lookup

```
nslookup www.example.com 192.168.255.53
# will return
Server: another.domain.com
Address: 192.168.255.53

Name: www.example.com
Address: 192.168.2.80
```

Returns the A record for www.example.com using the DNS server at 192.168.255.53. The command format allows either an IP or a name so the above command could have been written as:

```
nslookup www.example.com another.domain.com
```

Interactive Format

Interactive format (format 3 and 4 above) provides a single prompt (>) and allows any command line option to be entered. To terminate interactive mode you can use CTRL-C (Windows and *nix) or CTRL-D (*nix only) or **exit** (Windows and *nix).

Options

nslookup provides a dizzy number of options that vary its processing. Some of these options are only available in interactive mode. The Windows version adds a couple of commands. In each case **Mode** defines B = Interactive and command line format, I = Interactive only, C = command line only, W = Windows only. Multiple options can be specified with a single command.

option	params	mode	processing
d	-	C	Lists information for the domain. Gives SOA record and NS record

			details.
ls	[opt] domain	I	list all the information for the target domain. Takes the optional extensions > or >> filename to output to a file for subsequent processing. The options supported are: -a lists aliases (CNAME) in the domain (synonym for -t CNAME) -d The default behaviour. Lists all records in the domain (synonym for -t ANY) -h Lists all information records in the domain (synonym for -t HINFO) -s Lists all well known service records in the domain (synonym for -t WKS) -t List the specific record type in the domain e.g. -t A
lserver	dns	I	sets the dns for subsequent commands. May be either a name or an IP. The name or IP is looked up using the original default dns (before any server or lserver commands were issued).
root	root-dns	B	changes to root server used in various commands.
server	dns	I	sets the dns for subsequent commands. May be either a name or an IP. The name or IP is looked up using the current default dns. The default server is defined in /etc/resolv.conf for *nix systems and network properties for Windows systems.

options which work with 'set' in interactive mode

In interactive mode these options are preceded with **set** and operate until changed with another set directive. In command line mode they are preceded with **-** and operate on a single command. In a number of cases a short form is also provided.

all	-	B	displays a list of the default values used by nslookup, including the DNS server. Typical ouput
			`Set options:`
			`  nodebug`
			`  defname`
			`  search`
			`  recurse`
			`  nod2`
			`  novc`
			`  noignoretc`
			`  port=53`
			`  type=A`
			`  class=IN`
			`  timeout=2`
			`  retry=1`
			`  root=A.ROOT-SERVERS.NET.`
			`  domain=example.com`
			`  MSxfr [note: Windows only MS fast zone xfer]`
			`  IXFRversion=1 [note: Windows only incremental zone xfer]`
			`  srchlist=example.com`
			`Default Server:  ns1.example.com`
			`Address:  192.168.2.53`

class=	IN ANY CHAOS HESIOD	B	allows the class value to be set for all subsequent commands

domain=	domain-name	B	allows a base to be set for all subsequent searches e.g.
			`# assume default domain = example.com`
			`> set domain=example.org`
			`> www`
			`# returns results for www.example.org`

```
# but will handle full format
> mail.example.org
# returns correct result for mail.example.org
```

The default domain is defined in /etc/resolv.conf for *nix systems and network properties for Windows systems. Setting domain= will reset any previously defined **srchlist**.

[no]debug [no]deb	-	I	allows control over the debugging information - **debug** (short form **deb**) turns it on, **nodebug** (or **nodeb**) turns it off. The default is **nodebug**
[no]d2	-	I	enables/disable voluminous debugging information - **d2** turns it on, **nod2** turns it off. Default is **nod2**
[no]defname [no]def	-	I	controls whether a domain name (in either domain or srchlist) is added to a target which does not end with a dot i.e. is NOT a FQDN. See also search below for full behaviour description.
[no]ignoretc	-	I	controls if packet truncation errors are ignored (**ignoretc**) or whether they cause termination (**noignoretc** - default).
[no]msxfer	-	W	Controls use of MS Fast zone transfer. **msxfer** turns it on, **nomsxfer** (default) turns it off.
[no]recurse [no[rec]	-	B	Controls recursive behaviour. **recurse** (default) turns it on **norecurse** turns it off.
[no]vc	-	I	controls whether to use TCP (**vc**) or UDP (novc) - default is **novc**.
[no]search [no]sea	-	I	This parameter controls how the srchlist= value is used. **search** and **defname** are interrelated based on the following matrix for targets which are not FQDNs:

search	defname	add domain names from srchlist or until answer found
nosearch	defname	add domain name from domain
nosearch	nodefname	must be FQDN
search	nodefname	must be FQDN

In all cases the first result will terminate the command - you cannot use the srchlist to look up multiple targets. In general the srchlist is most useful with subdomains but can be used with different domains.

port=	port no.	B	changes the default port from the normal (53) to that specified by port no..
type= querytype=	ANY A CNAME HINFO MINFO MX NS PTR SOA TXT UINFO WKS	B	When using type= anything except A the following commands will only work on the domain root e.g.:

```
# enter interactive mode
nslookup
> set type=MX
> www.example.com
# fails with 'domain non-existent'
> example.com
# provides correct answers
```

ANY with a domain root name will return any DNS RR with a blank name (label) entry - these include NS and MX records and thus it provides a quick way to get useful domain info.

retry=	number	B	controls the number of retries that will be attempted. Default is 4.
root=	dns	B	controls the dns used in the root command. Default is typically f.root-server.net. (on *nix) and a.root-servers.net.) on windows.
srchlist=	dom1/dom2	I	allows setting of a searchlist (up to six names are allowed separated

by forward slash).

Examples - command line

```
# get mail records for a domain
nslookup -type=MX example.com
# list all the options being used and get host address
nslookup -all mail.example.com
# get SOA record using a specific DNS
nslookup -type=SOA example.com 192.168.23.53
```

Examples - interactive mode

```
# enter interactive mode and list default options
nslookup -all
>
# list all records in the domain
> ls example.com
# list all text records in domain
> ls -t TXT example.com
# set the base domain to be used for subsequent commands
> set domain=example.org
# find host
> mail
# returns mail.example.org
# exit interactive mode
> exit
```

10.3 DIG

dig is the current diagnostic DNS diagostic tool of preference but as noted above is not always widely available. You may still need to use nslookup.

Command Format

dig has both a command line and a batch mode (no interactive mode like **nslookup**). In general the command line of **dig** is more powerful than nslookup (even allowing multiple queries in a single line) and the batchmode makes running check files a breeze. **dig** offers a daunting array of options but the following are simple examples:

```
# get the A record for any record without a label
# but will always return the SOA record for the domain
dig example.com
# get the MX record for the domain
dig example.com mx
# get the A record for the host
dig www.example.com
# get all domain records if allowed
dig example.com axfr
# get all records with no label for the domain
dig example.com any
# typically returns SOA, NS, MX and domain SPF if defined
```

Generic Format

The following is the generic **dig** command format:

```
dig [@dns] domain [[-c ]q-type] [[-t ]q-class] [+q-opt] [-d-opt] [%comment]
```

Note: In general **dig** uses a mixture of positional/contextual arguments and **identified options** (i.e. identified with a option value @, -, +) to keep simple queries - simple! There are times when it necessary to disambiguate the **q-type** and **q-class** option and in this both can be specified in an **identified option format** (see examples).

Parameters in bone-chilling detail:

Parameter	Value	Description
dns	-	optional name or IP address (IPv4 or IPv6 format) of the DNS server to be used for the query. Default is defined in /etc/resolv.conf for *nix systems. If present must be preceded by commercial at (@) e.g. `dig @192.168.2.53 www.example.com`
domain	-	name or IP address (IPv4 or IPv6 format) of the target - may be a host or domain name depending on context (see examples).
q-type	a any axfr hinfo mx ns soa srv wks	Defines the type of record to return. May be optionally preceded with **-t** in the identified option format. Most values are self explanatory but to get a full listing of all the domain records use the **axfr** option. This feature may be disallowed by the allow-transfer BIND9 option in which case the command will fail with **Connection refused**.

q-class	in
	any
	hesiod
	chaos
		in is the default option. May be optionally preceded with **-c** in the identified option format. NB **any** is a valid option for both **q-type** and **q-class** and to ensure the correct value is used (to disambiguate in the jargon) always specify both **q-type** and **q-class** when using this format e.g.:
		`# this will get any record for class IN only`
		`dig example.com any`
		`# this will get any record for any class`
		`dig example.com any any`
		Alternatively you can use an **identified option format** with **-c** for q-class and **-t** for q- type. When the **identified option format** is used the parameter order

		not important e.g. `dig -c any -t any example.com` See **d-opt** below for **identified option format**
q-opt		All these options are preceded with a plus (+) and control how the resulting DNS query operates. Multiple values may appear in a single command. Many of the values are the same as nslookup. Many of the values have an abbreviation - its is shown in parenthesis after the command e.g. addit (ad). In this case **ad** is the abbreviation for **addit**.
	domain=name	Replaces the default domain name.
	[no]aaonly	Controls whether to use authoritative query only. Default = noaaonly.
	[no]addit	Controls whether to print additional information. Default = addit/ad.
	[no]answer	Controls whether to print answer section. Default = answer/an.
	[no]author	Controls whether to print authoritative section. Default = author/au.
	[no]cl	Controls whether to print class information. Default = nocl.
	[no]cmd	Controls whether to echo valid arguments. Default = cmd.
	[no]d2	Controls the voluminous diagnostic level. Default = nod2.
	[no]debug	Controls the diagnostic level. Default = nodebug.
	[no]defname	Controls substitution of default domain if no periods in domain name. Default = defname.
	[no]dnssec	Controls whether to set the DNSSEC OK bit in the OPT pseudo header. Default = nodnssec/nodn.
	[no]header	Print header flags. Default = header/he.
	[no]Header	print basic header. Default = Header/H.
	[no]ignore	Controls whether to ignore truncation errors. Default = noignore.
	[no]ko	Controls whether the virtual connection is kept open or not. Only valid with vc. Default = noko.
	[no]primary	Controls where to use or not the primary dns. Default = noprimary.
	[no]ques	Controls whether to print question section. Default = ques/qu.
	[no]qr	Controls whether to print outgoing query. Default = noqr.
	[no]recurse	Controls recursive query behaviour. Default = recurse.

	[no]reply	Controls whether to print a reply. Default = reply/rep.
	[no]search	Controls use of the srchlist (see explanation of relationship between **search** and **defname**). Default = search.
	[no]stats	Controls whether to display stats. Default = stats/st.
	[no]trunc	Controls whether to truncate origin from names. Default = trunc/tr.
	[no]ttlid	Controls whether to print TTL. Default = ttlid/tt.
	[no]vc	Controls whether to use TCP (**vc**) or UDP (**novc**). Default = novc.
	pfand=#	Bitwise AND print flags with # (octal/hex/decimal).
	pfdef	Set default print flags.
	pfmin	Set to minimal default print flags.
	pfor=#	Bitwise OR print flags with # (octal/hex/decimal).
	pfset=#	Set print flags as # (octal/hex/decimal).
	retry=num	Controls the number of query retries.
	time=secs	Controls the query timeout period. Default = 4 secs.
	time=secs	Controls the query timeout period. Default = 4 secs.
d-opt		These options control how **dig** operates and are preceded with a minus (-). Multiple options may appear in a single command line.
	-c	indicates a **q-class** argument follows (this is the **identified option format**) and can be used as a convenience or to disambiguate from the same **q-type options.**
	-envsav	save variables to the file defined by the environment variable LOCALDEF or DIG.env in the current working directory if LOCALDEF not set.
	-f filename	specifies a file containing batch commands. Any options specified on the command line will be in effect during the batch run i.e. they are global). Lines beginning with ';' or '#' or '\n' are ignored
	-k dir:key	Sign the key with TSIG **key** in **dir**.
	-p port	changes the port used for queries to **port (default is 53).**
	-P	causes a ping to be issued to the dns being used.
	-T secs	time in seconds between executing lines in a batch file (using option -f above)
	-t	indicates a **q-type** argument follows (this is the **identified argument format**).
	-x	specifies that inverse notation is being used i.e.:

```
# this will fail NXDOMAIN (not found)
dig 192.168.2.53
# instead use
dig -x 192.168.2.53
# OR if you are a masochist!
dig 53.2.168.192.in-addr.arpa ptr
```

Dig examples

Host Query

```
# simple host lookup - defaults to an A RR
dig www.example.com
# or could have been written as - order important
dig www.example.com a
# identified option format - order not important
dig -t a www.example.com
# use the dns at 192.168.2.224 for the query
dig @192.168.2.224 www.example.com a
# use the dns at ns1.example.com for the query
dig @ns1.example.com www.example.com a
# reverse map query - returns PTR RR
dig -x 192.168.2.224
```

Domain Query

```
# simple domain lookup - returns any A RR without a label
# even if none present will return the domain SOA RR
dig www.example.com
# quick domain lookup
# return all RR without labels - typically gets SOA, NS, MX and domain SPF if present
dig example.com any
# identified option format - order not important
dig -t any example.com
# use the dns at 192.168.2.224 for the query
dig @192.168.2.224 example.com any
# use the dns at ns1.another.com for the query
dig @ns1.another.com example.com a
```

Multiple Queries

You can issue multiple queries per command line - as long as each query is clearly identified (or disambiguated).

```
# multiple domain lookup - returns non-label RRs for both domains
dig example.com any another.com any
# multiple domain lookup - returns A RR for first and non-label RRs for second domains
dig example.com another.com any
# multiple domain lookup - returns non-label RRs for first domains and A RR for second
dig example.com any another.com
# if you start with one format you must be consistent - this fails on the second query
dig example.com -t any another.com any
# but this works
dig example.com -t any another.com -t any
# and yes this works
dig example.com any another.com any yetanother.com any
# and so does this
dig www.example.com www.another.com fred.yetanother.com
```

Chapter 15 DNS Messages

15.1 Overview

This section details the format of messages that pass between a Resolver and a DNS system. The really smart thing to do is install ethereal and let it to all the analysis for you. However if you are in de-bug mode then you may need this stuff. This where the Rocket Scientists wannabees hang out. Welcome.

Message formats are defined in RFC 1035.

The good news is that each message has the same generic format with 5 sections. This is the last good news.

Section	Meaning / Use
Section 1	Message Header
Section 2	The DNS question being asked
Section 3	The Resource Record(s) which answer the question
Section 4	The Resource Record(s) which point to the domain authority
Section 5	The Resource Record(s) which may hold additional information

Notes:

1. Unused sections are not present - determined by count values in the message header

15.2 The Message Header

Present in all messages. Never empty. Contains various flags and values which control the transaction. If you are not comfortable with bits, bytes and hex values take up origami or read this quick memory jogger. And while you are in this

receptive mode you may want remind yourself that bit numbering standards are a real mess.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Message ID															
QR	OPCODE				AA	TC	RD	RA	res1	res2	res3	RCODE			
QDCOUNT															
ANCOUNT															
NSCOUNT															
ARCOUNT															

The following table defines the field values above:

Section	Meaning / Use
Message ID	16 bit message ID supplied by the requestion (the questioner) and reflected back unchanged by the responder (answerer). Identifies the transaction.
QR	Query - Response bit. Set to 0 by the questioner (query) and to 1 in the response (answer).
OPCODE	Identifies the request/operation type. Currently assigned values are: **Value** / **Meaning / Use** 0 — QUERY. standard query. 1 — IQUERY. Inverse query. Optional support by DNS 2 — STATUS. DNS status request
AA	Authoritative Answer. Valid in responses only. Because of aliases multiple owners may exists so the AA bit corresponds to the name which matches the query name, OR the first owner name in the answer section.
TC	TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel. Set on all truncated messages except the last one.
RD	Recursion Desired - this bit may be set in a query and is copied into the response if recursion supported if rejected the response (answer) does not have this bit set. Recursive query support is optional.
RA	Recursion Available - this bit is valid in a response (answer) and denotes whether recursive query support is available (1) or not (0) in the name server.
RCODE	Identifies the response type to the query. Ignored on a request (question). Currently assigned values: **Value** **Meaning / Use**

	0 No error condition. 1 Format error - The name server was unable to interpret the query. 2 Server failure - The name server was unable to process this query due to a problem with the name server. 3 Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist. 4 Not Implemented - The name server does not support the requested kind of query. 5 Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data.
QDCOUNT	Unsigned 16 bit integer specifying the number of entries in the question section.
ANCOUNT	Unsigned 16 bit integer specifying the number of resource records in the answer section. May be 0 in which case no answer record is present in the message.
NSCOUNT	Unsigned 16 bit integer specifying the number of name server resource records in the authority section. May be 0 in which case no authority record(s) is(are) present in the message.
ARCOUNT	Unsigned 16 bit integer specifying the number of resource records in the additional records section. May be 0 in which case no addtional record(s) is(are) present in the message.

Notes:

1.

15.3 The DNS Question

While it is normal to have only one question per message, it is permissible to have any number defined by QDCOUNT each question has the same format as defined below:

Field Name	Meaning/Use
QNAME	The domain name being queries
QTYPE	The resource records being requested
QCLASS	The Resource Record(s) class being requested e.g. internet, chaos etc.

Each field has the following format:

<table>
<tr><th>Name</th><th>Meaning / Use</th></tr>
<tr><td>QNAME</td><td>The name being queried will depend upon the QTYPE (below) e.g. a request for an A record will require a host part i.e. www.mydomain.com an MX query will obviously only use a domain name i.e. mydomain.com The name being queried is split into labels by removing the separating dots. Each label is represented as a length/data pair as follows:
<table>
<tr><th>Value</th><th>Meaning / Use</th></tr>
<tr><td>no. of chars</td><td>single octet defining the number of characters in the label which follows. The top two bits of this number must be 00 (indicates the label format is being used) which gives a maximum domain name length of 63 bytes (octets). If we assume this is comprised of a TLD + domain and the TLD (and its accompanying dot) can have a maximum length 4 bytes this gives the well-known maximum domain name length of 59 bytes (octets). Well that was true until we had .museum and all the other new TLDs so each domain name limit is actually TLD specific. Another global truth falls by the wayside. A value of zero indicates the end of the name field.</td></tr>
<tr><td>domain name</td><td>A string containing the characters in the label.</td></tr>
</table>
Wow. To illustrate the above we'll use two examples:
<pre>
// assume an MX query with a name of mydomain.com
// the hex representation is
08 6D 79 64 6F 6D 61 69 6E 03 63 6F 6D 00
// printable
 ! m y d o m a i n ! c o m !
// note ! = unprintable

// assume an A query with a name of www.mydomain.com
// the hex representation is
03 77 77 77 08 6D 79 64 6F 6D 61 69 6E 03 63 6F 6D 00
// printable
 ! w w w ! m y d o m a i n ! c o m !
// note ! = unprintable
</pre></td></tr>
<tr><td>QTYPE</td><td>Unsigned 16 bit value. The resource records being requested. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values:
<table>
<tr><th>Value</th><th>Meaning / Use</th></tr>
<tr><td>x'0001 (1)</td><td>Requests the A record for the domain name</td></tr>
<tr><td>x'0002 (2)</td><td>Requests the NS record(s) for the domain name</td></tr>
<tr><td>x'0005 (5)</td><td>Requests the CNAME record(s) for the domain name</td></tr>
<tr><td>x'0006 (6)</td><td>Requests the SOA record(s) for the domain name</td></tr>
<tr><td>x'000B (11)</td><td>Requests the WKS record(s) for the domain name</td></tr>
<tr><td>x'000C (12)</td><td>Requests the PTR record(s) for the domain name</td></tr>
<tr><td>x'000F (15)</td><td>Requests the MX record(s) for the domain name</td></tr>
<tr><td>x'0021 (33)</td><td>Requests the SRV record(s) for the domain name</td></tr>
<tr><td>x'0026 (38)</td><td>Requests the A6 record(s) for the domain name</td></tr>
<tr><td>x'00FF (255)</td><td>Requests ANY resource record (typically wants SOA, MX, NS and MX)</td></tr>
</table></td></tr>
</table>

QCLASS	Unsigned 16 bit value. The CLASS of resource records being requested e.g. Internet, CHAOS etc. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values: **Value** — **Meaning/Use** x'0001 (1) — IN or Internet

15.4 The DNS Answer

The Answer, Authority and Additional Section all comprise RRs and hence share the same format. The section the record appears in determines its type e.g. an A RR can appear in the Answer or Additional section. So far this stuff has been relatively straightforward if messy - take a deep breath before reading on. The format of these records is:

Field Name	Meaning/Use
NAME	The name being returned e.g. www or ns1.example.net If the name is in the same domain as the question then typically only the host part (label) is returned, if not then a FQDN is returned.
TYPE	The RR type e.g. SOA or AAAA
CLASS	The RR class e.g. Internet, Chaos etc.
TTL	The TTL in seconds of the RR e.g. 2800
RLENGTH	The length of RR specific data in octets e.g. 27
RDATA	The RR specific data (see Binary RR Formats below) whose length is defined by RDLENGTH e.g. 192.168.254.2

The various fields have the following meanings:

Name	Meaning/Use
NAME	This name reflects the QNAME of the question i.e. any may take one of TWO formats. The first format is the **label** format defined for QNAME above. The second format is a pointer (in the interests of data compression which to fair to the original authors was far more important then than now). A pointer is an unsigned 16-bit value with the following format (the top two bits of 11 indicate the pointer format):

<table>
<tr><td></td><td>
<table>
<tr><td>1</td><td>1</td><td>The offset in octets (bytes) from the start of the whole message. Must point to a label format record to derive name length.</td></tr>
</table>

Note: Pointers, if used, terminate names. The name field may consist of a label (or sequence of labels) terminated with a zero length record OR a single pointer OR a label (or label sequence) terminated with a</td></tr>
<tr><td>TYPE</td><td>Unsigned 16 bit value. The resource record types - determines the content of the RDATA field. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values:

<table>
<tr><th>Value</th><th>Meaning / Use</th></tr>
<tr><td>x'0001 (1)</td><td>An A record for the domain name</td></tr>
<tr><td>x'0002 (2)</td><td>A NS record(for the domain name</td></tr>
<tr><td>x'0005 (5)</td><td>A CNAME record for the domain name</td></tr>
<tr><td>x'0006 (6)</td><td>A SOA record for the domain name</td></tr>
<tr><td>x'000B (11)</td><td>A WKS record(s) for the domain name</td></tr>
<tr><td>x'000C (12)</td><td>A PTR record(s) for the domain name</td></tr>
<tr><td>x'000F (15)</td><td>A MX record for the domain name</td></tr>
<tr><td>x'0021 (33)</td><td>A SRV record(s) for the domain name</td></tr>
<tr><td>x'0026 (38)</td><td>An A6 record(s) for the domain name</td></tr>
</table></td></tr>
<tr><td>CLASS</td><td>Unsigned 16 bit value. The CLASS of resource records being requested e.g. Internet, CHAOS etc. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values:

<table>
<tr><th>Value</th><th>Meaning / Use</th></tr>
<tr><td>x'0001 (1)</td><td>IN or Internet</td></tr>
</table></td></tr>
<tr><td>TTL</td><td>Unsigned 32 bit value. The time in seconds that the record may be cached. A value of 0 indicates the record should not be cached.</td></tr>
<tr><td>RDLENGTH</td><td>Unsigned 16-bit value that defines the length in bytes (octets) of the RDATA record.</td></tr>
<tr><td>RDATA</td><td>Each (or rather most) resource record types have a specific RDATA format which reflect their resource record format as defined below:

SOA

<table>
<tr><th>Value</th><th>Meaning / Use</th></tr>
<tr><td>Primary NS</td><td>Variable length. The name of the Primary Master for the domain. May be a label, pointer or any combination.</td></tr>
</table></td></tr>
</table>

Admin MB	Variable length. The administrator's mailbox. May be a label, pointer or any combination.
Serial Number	Unsigned 32-bit integer.
Refresh interval	Unsigned 32-bit integer.
Retry Interval	Unsigned 32-bit integer.
Expiration Limit	Unsigned 32-bit integer.
Minimum TTL	Unsigned 32-bit integer.

MX

Value	Meaning / Use
Preference	Unsigned 16-bit integer.
Mail Exchanger	The name host name that provides the service. May be a label, pointer or any combination.

A

Value	Meaning / Use
IP Address	Unsigned 32-bit value representing the IP address

PTR, NS

Value	Meaning / Use
Name	The host name that represents the supplied IP address (in the case of a PTR) or the NS name for the supplied domain (in the case of NS). May be a label, pointer or any combination.

15.5 Domain Authority

You will be delighted to know that authority records have exactly the same format as Answer records it is simply their position in an authority section that determines they are authority records (and that they will be of TYPE NS).

15.6 Additional Information

You will be delighted to know that additional records have exactly the same format as Answer records it is simply their position in an additional section that determines they are additional records.

Memory jogger - Binary, decimal and hexadecimal

The contents of an 8 bit byte (an octet) may be expressed in decimal (base 10), binary (base 2) or hexadecimal (base 16 - 0-9, A-F) as follows:

Decimal	Hexadecimal	Binary
0	00	0000 0000
65	41	0100 0001
187	BB	1011 1011
255	FF	1111 1111

To convert a dotted decimal IP e.g. 192.168.0.5 to hexadecimal, take each dotted decimal value and convert it using a hex calculator (standard windows calculator in scientific mode will do the job). This will yield C0.A8.00.05 for our example above.

Bit numbering

Bit numbering can be very confusing with various standard bodies adopting different conventions. The following are all valid, and used, bit numbering **conventions** for describing an 8 bit byte (an octet).

Memory contents	0	0	0	0	0	0	0	0
Bit numbering conventions								
Left to right base 0 (IETF)	0	1	2	3	4	5	6	7
Left to right base 1	1	2	3	4	5	6	7	8
Right to left base 1 (ITU)	8	7	6	5	4	3	2	1
Power of 2	7	6	5	4	3	2	1	0

Always check what convention is used on any specification. The convention used by the IETF is a LEFT to RIGHT number starting from (base) ZERO. Many (but not all) C compliers allocate bits in a field using this convention.

Time-to-Live (TTL) Values

The $TTL directive is defined in RFC 2308.

TTL in the DNS context defines the duration in seconds that the record may be cached. Zero indicates the record should not be cached. **Note:** RFC 1912 cautions that 0 = no caching is not widely implemented so make no assumptions.

The default TTL for the zone is defined in BIND9 by the $TTL directive which must appear at the beginning of the zone file i.e. before any RR to which it will apply. This $TTL is used for any Resource Record which does not explicitly set the 'ttl' field.

The TTL field is defined to be an unsigned 32 bit value with a valid range from 0 to 2147483647 (clarified in RFC 2181) - which is a long time! - somewhere on the other side of 68 years.

The $TTL field may take any time value.

In BIND 8 the SOA record (minimum parameter) was used to define the zone default TTL value. In BIND 9 the SOA 'minimum' parameter is used as the negative (NXDOMAIN) caching time (defined in RFC 2308).

RFC 1912 recommends that the $TTL value be set to 1 day or longer and that certain RRs which rarely change, such as the MX records for the domain, use an explicit TTL value to set even longer values such as 2 to 3 weeks. The value of this field is a balance between how frequently you think the DNS records will change vs load on the DNS server. In the example below the $TTL value of 2d (2 days) indicates that any change may not be fully propagated for 48 hours, equally caching DNS servers will require to re-read the RRs from your DNS every 48 hours which can be a non-trivial load. Many users will set this value to say 2w (2 weeks) in normal operation then prior to planned changes will reduce the value to say 1d or 12h, until the change has stabilized then restore the value to 2w.

Example

```
; example.com zone file fragment
$TTL 2d ; zone default
      3w   IN      MX  10 192.168.254.2  ; overrides default
joe   3h   IN      A      192.168.254.3  ; overrides default
www        IN      A      192.168.254.3  ; uses zone default = 2 days
```

ORIGIN and @ Substitution

The symbol @ is used in BIND to denote 'zone root'. The value substituted for @ is either:

- The last $ORIGIN directive encountered in the file.
- OR
- If no $ORIGIN directive - BIND uses the value of the zone in named.conf file e.g.

```
// named.conf file fragment

zone "example.com" in{
        type master;
        file "pri.example.com";
};
```

example.com will replace @.

-

Example

```
; example.com zone file fragment
....
@          IN      NS     ns1.mydomain.com.
; ns1.example.com services example.com
....
$ORIGIN uk.example.com.
@          IN      NS     ns2.example.com.
; ns2.example.com services uk.example.com
```

Reverse Look-up Problem

If you define two A Records with the same IP address then when you come to define the Reverse Mapping (look-up) file you can only map a single name to the IP address. Increasingly mail systems as part of an anti-spam strategy and others may perform a dual look-up, IP to name then name to IP. A Reverse Mapping query will only return the right result for joe in the example below not for www. In this simple case use of a CNMAE record for www could fix the problem.

Example

```
; zone file fragment
joe        IN      A      192.168.254.3  ; joe & www = same ip
www        IN      A      192.168.254.3

; reverse in-arpa file fragment
.3         IN      PTR    joe.mydomain.com.
```

If you define multiple A Records for load balancing or round robin strategy the same problem arises. In a dual look-up, IP to name then name to IP, system this will only work for mail in the example below not for any others.

```
; zone file fragment
mail        IN      A       192.168.254.3  ; round robin
            IN      A       192.168.254.4
            IN      A       192.168.254.5

; reverse in-addr.arpa file fragment
.3          IN      PTR     mail.mydomain.com.
.4          IN      PTR     mail1.mydomain.com.
.5          IN      PTR     mail2.mydomain.com.
```

The DOT in a Zone File

Sometimes you need it sometimes you don't. At first glance, and even at the fourth glance, it seems confusing.

It is not. The rule is simple.

If there is a dot at the end of a name in a resource record or directive, the name is 'qualified' and if it contains the whole name including the host then it is a Fully Qualified Domain Name - FQDN. The value as it appears in the record is used **unchanged**.

If there is NO dot at the end of the name (a.k.a. 'label' in DNS jargon), the name is 'unqualified' and DNS software adds the 'zone name' from the named.conf file for this zone OR the value of the last $ORIGIN statement. The fragment below illustrates this using A records and CNAME records.

```
; zone file fragment for mydomain.com
; the named.conf file contains 'zone "mydomain.com"'
; there is no $ORIGIN statement
; line below is expanded to joe.mydomain.com
joe               IN      A       192.168.254.3
; next line www.mydomain.com aliased to joe.mydomain.com
www               IN      CNAME   joe
; next line is functionally the same as line above
www.mydomain.com. IN      CNAME   joe.mydomain.com.
; and so is this line
www               IN      CNAME   joe.mydomain.com.
; this is record defaults to mydomain.com
                  IN      A       192.168.254.3
```

BIND Time formats

BIND allows a number of time formats in combinations. Formats allowed are (case insensitive):

- #s = seconds = # x 1 seconds (really!)
- #m = minutes = # x 60 seconds
- #h = hours = # x 3600 seconds
- #d = day = # x 86400 seconds
- #w = week = # x 604800 seconds

Examples

```
; foo.com zone file fragment
; common origin (@) format
@           IN      SOA   ns.foo.com. root.foo.com. (
                2003080800 ; serial number
                10800s     ; refresh =  3 hours
                15M        ; update retry = 15 minutes
                3W12h      ; expiry = 3 weeks + 12 hours
                2h20M      ; minimum = 2 hours + 20 minutes
                )
www 12H15m  IN     A      10.0.5.17 ; A record TTL value
```

Zones and Zone files

A 'zone' is convenient short-hand for that part of the domain name for which we are configuring the DNS server (e.g. BIND) and is always an entity for which we are authoritative.

Assume we have a 'Domain Name' of mydomain.com. This is comprised of a domain-name (mydomain) and a gTLD name (com). The zone in this case is 'mydomain.com'. If we we have a sub-domain which has been delegated to us called us.mydomain.com then the zone is 'us.mydomain.com'.

Zones are described in zone files (sometimes called master files) (normally located in /var/named) which can contain Directives (used by the DNS software e.g. BIND) and Resource Records which describe the characteristics of the zone and individual hosts and services within the zone. Both Directives and Resource records are a standard defined by RFC 1035 so should be read by any self-respecting DNS server software. The single exception to this is the BIND-specific $GENERATE directive. So if you think you will change DNS servers don't use $GENERATE.

Example Zone File

```
foo.com.      IN      SOA   ns1.foo.com. root.foo.com. (
                              2003080800 ; se = serial number
                              3h         ; ref = refresh
                              15m        ; ret = update retry
                              3w         ; ex = expiry
```

```
                                3h         ; min = minimum
                                )
                IN      NS      ns1.foo.com.
                IN      NS      ns2.foo.com.
                IN      MX  10  mail.anotherdomain.com.
joe             IN      A       192.168.254.3
www             IN      CNAME   joe
```

DNS Queries

The major task carried out by a DNS server is to respond to queries (questions) from a local or remote resolver or other DNS acting on behalf of a resolver - a query would be something like 'what is the IP address of host=fred in domain=mydomain.com'. There are three types of queries that DNS support:

1. A recursive query - the real answer to the question is always returned. DNS servers are not required to support recursive queries.
2. An Iterative (or non-recursive) query - where the real answer MAY be returned. All DNS servers must support Iterative queries.
3. A Inverse query - where the user wants to know the domain name given a resource record.

Note: The process called Reverse Mapping does not use Inverse queries but instead uses recursive and non-recursive queries with the special domain name IN-ADDR.ARPA.

Historically reverse IP mapping was non-mandatory. Many systems however now use reverse mapping for security and simple authentication schemes and is its proper implementation and maintenance is now essential.

Resolvers

The generic term **resolver** defines a set of functions supplied as part of the standard C network/socket libraries (i.e. glibc6 in *nix systems) or supplied as part of a package (e.g. BIND). These functions are used by applications to answer questions such as 'what is the IP address of this host'. The most common method to invoke such resolver services, used by your browser among many other applications, is to use the POSIX socket functions 'gethostbyname' (or 'getaddrinfo' for sock2) for name to IP and 'gethostbyaddr' (replaced by 'getnameinfo' in sock2) for IP to name.

Resolvers are quite complicated and are defined to be capable of following **referrals** (they can work with systems that do not support recursive queries. However almost all resolvers (both Windows and *nix) are **stub** resolvers. A **stub** resolver is a minimal resolver which will only work with a DNS that does support **recursive** queries i.e. it cannot follow referrals. Some newer Windows systems (Windows 2K and XP) provide what is called a **caching resolver. This resolver is a stub**

resolver but does maintain a cache of responses to minimize network access and increase performance.

There are a number of ways your system can resolve a name and the actual order will vary based on your configuration:

1. **If you are using a *nix system with the GNU glibc libraries the order of lookup is determined by the 'hosts' entry in the /etc/nsswitch.conf file which will read something like:**

```
hosts files nisplus dns
```

   Indicating look at /etc/hosts, then use NIS (Network Information Systems), then DNS (via resolv.conf)

2. **If you are using a *nix system with the older GNU libc libraries the order of lookup is determined by the 'order' entry in the /etc/host.conf file which will read something like:**

```
order hosts,bind
```

   Indicating look at /etc/hosts then DNS (using resolv.conf)

3. **If you are using a windows system the order is:**
   1. **look in hosts (windows\system32\drivers\etc\hosts)**
   2. **use the DNS entries in the tcp/ip network definition.**
4. **FreeBSD does not install /etc/nsswitch.conf by default so NIS assumes a resolution order as if the following 'hosts' line was present in nsswitch.conf:**

```
hosts dns files
```

   Which means use DNS (via resolve.conf) then /etc/hosts.

If you want to know more about resolvers on *nix systems read Chapter 6 from the Linux Network Administrators Guide.

DNS Referrals

The term **referral** indicates a response to a query which does not contain an answer section (it is empty) but which contains one or more authoritative name servers (in the Domain Authority section) that are closer to the required query question. The response will typically (always from the root and TLD servers) contain in the Additional Information section the IP addresses (the A or glue records) of the supplied servers. A query to the root-servers for the IP of fred.example.com will result in the following DNS transactions:

1. Root-server returns a *referral* to the gTLD servers for .com (list of servers in Authority section and IP addresses in Additional Information section of response
2. gTLD server returns a *referral* to the name servers for example.com (list of servers in Authority section and IP addresses in Additional Information section of response

Resource Records

Resource Records are defined by RFC 1035 (and augmented by others). Resource Records describe global properties of a zone and the hosts or services that are part of the zone. They are described in detail in Chapter 8. Resource Records have a binary format, used internally by DNS software and when sent across a network e.g. zone updates, and a text format which is used in zone files.

Resource Records include SOA Record, NS Records, A Records, CNAME Records, PTR Records, MX Records.

Example of Resource Records in a Zone File

```
foo.com.      IN      SOA   ns.joe.com. root.foo.com. (
                                2003080800 ; se = serial number
                                3h         ; ref = refresh
                                15m        ; ret = update retry
                                3w         ; ex = expiry
                                3h         ; min = minimum
                                )
              IN      NS      ns1.foo.com.
              IN      MX  10  mail.anotherdomain.com.
joe           IN      A       192.168.254.3
www           IN      CNAME   joe
```

BIND: named.conf file

BIND operational functionality is controlled by a file called named.conf normally /etc/named.conf (Linux) or /etc/namedb/named.conf (BSD's).

The named.conf file declares both global properties for BIND and the zones for which the configuration is a slave or master. Each zone referenced requires a zone file. By default zone files are located in /var/named but this can be changed using the 'directory' parameter in named.conf.

Zone File Directives

RFC 1035 defines a number of directives for use in zone files. These are summarised below:

- $TTL - Mandatory as first entry in a BIND 9 zone file. Defines the default TTL (or cache life) for any Resource Record which does not contain one.
- $ORIGIN - changes the 'zone name' which is added to any 'unqualified' name. The dot is optional at the end of a name appearing in an ORIGIN directive.
- $INCLUDE - allows the contents of a file to be inserted into the zone file. The critical point to note is that if an INCLUDE'd file contains an $ORIGIN statement it's scope (is valid) for the INCLUDE'd file only, the original values of ORIGIN and 'zone name' are restored once the INCLUDE'd file has been fully processed.

BIND additionally provides $GENERATE, a non-standard directive to simplify generation of reverse delegations in in-addr.arpa files. If you think you may want to change DNS software do not use $GENERATE.

BIND statement layout variations

BIND is very picky about opening and closing brackets/braces, semicolons and all the other separators defined in the formal statement 'grammars. The literature contains various ways to layout statements. The variations are simply attempts by the authors to minimise the potential for errors, they have no other significance. Use any method you feel comfortable with. The following zone statement layouts are all equivalent and acceptable to BIND.

```
// dense single line
zone "mydomain.com" {type slave; file "sec.mydomain.com"; masters
{10.0.0.1;};};
// lots of lines
zone "mydomain.com" {
   type slave;
   file "sec.mydomain.com";
   masters {10.0.0.1;};
   };
// spot the difference
zone "mydomain.com" {
   type slave;
   file "sec.mydomain.com";
   masters {10.0.0.1;}; };
```

Classless Routing Overview

This is a simplified description of classless routing principles. For more information.

Classless routing, defined in RFC 1817, allows for the routing of non-octet boundary subnets and greatly increases the usage of the existing IPv4 address space. Prior to Classless routing only Class boundary routing was supported e.g.:

```
CLASS A address e.g. 10.0.0.0 subnet mask 255.0.0.0
CLASS B address e.g. 172.16.0.0 subnet mask 255.255.0.0
CLASS C address e.g. 192.168.0.0 subnet mask 255.255.255.0
```

Class C address ranges are used for the following examples but the principles apply to all address ranges.

Classless routing, in this context, describes the process of splitting and routing the Class C space into multiple subnets. Assume we want to allocate and route a 32 address subnet, starting at address 64, in the Class C address 192.168.23.0. This could be written as:

```
192.168.23.64 netmask 255.255.255.224
```

To simplify the process of writing this, a short-form (sometimes called the 'slash' form officially an **IP Prefix**) '/x' is used, where x defines the number of contiguous bits in the subnet mask. So the above could be written as:

```
192.168.23.64/26 (26 bits in the subnet mask 255.255.255.224)
```

Remember that the first address in the subnet is reserved for multi-casting and the last for broadcasting - you always lose two addresses in a subnet.

The following table shows the relationship between the 'slash' form and the subnet mask - starting with a base address of 0:

```
192.168.23.0/24 (subnet mask 255.255.255.0)
192.168.23.0/24 (256 IPs - subnet mask 255.255.255.0)
192.168.23.0/25 (128 IPs - subnet mask 255.255.255.128)
192.168.23.128/26 (64 IPs - subnet mask 255.255.255.192)
192.168.23.192/27 (32 IPs - subnet mask 255.255.255.224)
192.168.23.224/28 (16 IPs - subnet mask 255.255.255.240)
192.168.23.240/29 (8 IPs - subnet mask 255.255.255.248)
192.168.23.248/30 (4 IPs - subnet mask 255.255.255.252)
```

Valid Names and Labels

Host Names (or 'labels' in DNS jargon) were traditionally defined by RFC 952 and RFC 1123 and may be composed of the following valid characters.

```
A to Z ; upper case characters
a to z ; lower case characters
0 to 9 ; numeric characters 0 to 9
-      ; dash
```

The rules say:

1. A host name (label) can start or end with a letter or a number
2. A host name (label) MUST NOT start or end with a '-' (dash)

3. A host name (label) MUST NOT consist of all numeric values
4. A host name (label) can be up to 63 characters

RFC 2181 significantly liberalized the valid character set including the use of "_" (underscore) essentially saying that anything goes and its up to the client to validate in context. If you want to be safe stick with the rules above if you **need** the expanded capabilities (e.g. SRV RRs) use them. However you are taking a risk that one vital system will not talk to you at 3AM in the morning due to an upgrade!

The named.conf statement check-names allows control over the names accepted.

Domain names are defined to be case insensitive (essentially so you don't have to register every possible variant of your domain name) but the rule is that case should be **preserved** since this may change in the future. Host names **seem** to obey the same rules - essentially as an artifact of subdomains.

Host Name Examples

```
; for clarity we show a host name in A records
www         IN      A       192.168.0.3 ; valid
wWw         IN      A       192.168.0.3 ; valid
my www      IN      A       192.168.0.3 ; invalid
my-www      IN      A       192.168.0.3 ; valid
my_www      IN      A       192.168.0.3 ; invalid but may work
3www        IN      A       192.168.0.3 ; valid
-www        IN      A       192.168.0.3 ; invalid
5512        IN      A       192.168.0.3 ; invalid
host-5512   IN      A       192.168.0.3 ; valid
@www        IN      A       192.168.0.3 ; invalid
```