                  DOMAIN NAMES - CONCEPTS AND FACILITIES

1. STATUS OF THIS MEMO

This RFC is an introduction to the Domain Name System (DNS), and omits
many details which can be found in a companion RFC, "Domain Names -
Implementation and Specification" [RFC-1035]. That RFC assumes that the
reader is familiar with the concepts discussed in this memo.

A subset of DNS functions and data types constitute an official
protocol. The official protocol includes standard queries and their
responses and most of the Internet class data formats (e.g., host
addresses).

However, the domain system is intentionally extensible. Researchers are
continuously proposing, implementing and experimenting with new data
types, query types, classes, functions, etc. Thus while the components
of the official protocol are expected to stay essentially unchanged and
operate as a production service, experimental behavior should always be
expected in extensions beyond the official protocol. Experimental or
obsolete features are clearly marked in these RFCs, and such information
should be used with caution.

The reader is especially cautioned not to depend on the values which
appear in examples to be current or complete, since their purpose is
primarily pedagogical. Distribution of this memo is unlimited.

2. INTRODUCTION

This RFC introduces domain style names, their use for Internet mail and
host address support, and the protocols and servers used to implement
domain name facilities.

2.1. The history of domain names

The impetus for the development of the domain system was growth in the
Internet:

   - Host name to address mappings were maintained by the Network
     Information Center (NIC) in a single file (HOSTS.TXT) which
     was FTPed by all hosts [RFC-952, RFC-953]. The total network

     bandwidth consumed in distributing a new version by this
     scheme is proportional to the square of the number of hosts in
     the network, and even when multiple levels of FTP are used,
     the outgoing FTP load on the NIC host is considerable.
     Explosive growth in the number of hosts didn't bode well for
     the future.

   - The network population was also changing in character. The
     timeshared hosts that made up the original ARPANET were being
     replaced with local networks of workstations. Local
     organizations were administering their own names and
     addresses, but had to wait for the NIC to change HOSTS.TXT to
     make changes visible to the Internet at large. Organizations
     also wanted some local structure on the name space.

   - The applications on the Internet were getting more
     sophisticated and creating a need for general purpose name
     service.

The result was several ideas about name spaces and their management
[IEN-116, RFC-799, RFC-819, RFC-830]. The proposals varied, but a

using "." as the character to mark the boundary between hierarchy
levels. A design using a distributed database and generalized resources
was described in [RFC-882, RFC-883]. Based on experience with several
implementations, the system evolved into the scheme described in this
memo.

The terms "domain" or "domain name" are used in many contexts beyond the
DNS described here. Very often, the term domain name is used to refer
to a name with structure indicated by dots, but no relation to the DNS.
This is particularly true in mail addressing [Quarterman 86].

2.2. DNS design goals

The design goals of the DNS influence its structure. They are:

   - The primary goal is a consistent name space which will be used
     for referring to resources. In order to avoid the problems
     caused by ad hoc encodings, names should not be required to
     contain network identifiers, addresses, routes, or similar
     information as part of the name.

   - The sheer size of the database and frequency of updates
     suggest that it must be maintained in a distributed manner,
     with local caching to improve performance. Approaches that

     attempt to collect a consistent copy of the entire database
     will become more and more expensive and difficult, and hence
     should be avoided. The same principle holds for the structure
     of the name space, and in particular mechanisms for creating
     and deleting names; these should also be distributed.

   - Where there tradeoffs between the cost of acquiring data, the
     speed of updates, and the accuracy of caches, the source of
     the data should control the tradeoff.

   - The costs of implementing such a facility dictate that it be
     generally useful, and not restricted to a single application.
     We should be able to use names to retrieve host addresses,
     mailbox data, and other as yet undetermined information. All
     data associated with a name is tagged with a type, and queries
     can be limited to a single type.

   - Because we want the name space to be useful in dissimilar
     networks and applications, we provide the ability to use the
     same name space with different protocol families or
     management. For example, host address formats differ between
     protocols, though all protocols have the notion of address.
     The DNS tags all data with a class as well as the type, so
     that we can allow parallel use of different formats for data
     of type address.

   - We want name server transactions to be independent of the
     communications system that carries them. Some systems may
     wish to use datagrams for queries and responses, and only
     establish virtual circuits for transactions that need the
     reliability (e.g., database updates, long transactions); other
     systems will use virtual circuits exclusively.

   - The system should be useful across a wide spectrum of host
     capabilities. Both personal computers and large timeshared
     hosts should be able to use the system, though perhaps in
     different ways.

2.3. Assumptions about usage

The organization of the domain system derives from some assumptions
about the needs and usage patterns of its user community and is designed
to avoid many of the the complicated problems found in general purpose
database systems.

The assumptions are:

     to the number of hosts using the system, but will eventually
     grow to be proportional to the number of users on those hosts
     as mailboxes and other information are added to the domain
     system.

   - Most of the data in the system will change very slowly (e.g.,
     mailbox bindings, host addresses), but that the system should
     be able to deal with subsets that change more rapidly (on the
     order of seconds or minutes).

   - The administrative boundaries used to distribute
     responsibility for the database will usually correspond to
     organizations that have one or more hosts. Each organization
     that has responsibility for a particular set of domains will
     provide redundant name servers, either on the organization's
     own hosts or other hosts that the organization arranges to
     use.

   - Clients of the domain system should be able to identify
     trusted name servers they prefer to use before accepting
     referrals to name servers outside of this "trusted" set.

   - Access to information is more critical than instantaneous
     updates or guarantees of consistency. Hence the update
     process allows updates to percolate out through the users of
     the domain system rather than guaranteeing that all copies are
     simultaneously updated. When updates are unavailable due to
     network or host failure, the usual course is to believe old
     information while continuing efforts to update it. The
     general model is that copies are distributed with timeouts for
     refreshing. The distributor sets the timeout value and the
     recipient of the distribution is responsible for performing
     the refresh. In special situations, very short intervals can
     be specified, or the owner can prohibit copies.

   - In any system that has a distributed database, a particular
     name server may be presented with a query that can only be
     answered by some other server. The two general approaches to
     dealing with this problem are "recursive", in which the first
     server pursues the query for the client at another server, and
     "iterative", in which the server refers the client to another
     server and lets the client pursue the query. Both approaches
     have advantages and disadvantages, but the iterative approach
     is preferred for the datagram style of access. The domain
     system requires implementation of the iterative approach, but
     allows the recursive approach as an option.

The domain system assumes that all data originates in master files
scattered through the hosts that use the domain system. These master
files are updated by local system administrators. Master files are text
files that are read by a local name server, and hence become available
through the name servers to users of the domain system. The user
programs access name servers through standard programs called resolvers.

The standard format of master files allows them to be exchanged between
hosts (via FTP, mail, or some other mechanism); this facility is useful
when an organization wants a domain, but doesn't want to support a name
server. The organization can maintain the master files locally using a
text editor, transfer them to a foreign host which runs a name server,
and then arrange with the system administrator of the name server to get
the files loaded.

Each host's name servers and resolvers are configured by a local system

non-local master files are to be loaded from foreign servers. The name
server uses the master files or copies to load its zones. For
resolvers, the configuration data identifies the name servers which
should be the primary sources of information.

The domain system defines procedures for accessing the data and for
referrals to other name servers. The domain system also defines
procedures for caching retrieved data and for periodic refreshing of
data defined by the system administrator.

The system administrators provide:

   - The definition of zone boundaries.

   - Master files of data.

   - Updates to master files.

   - Statements of the refresh policies desired.

The domain system provides:

   - Standard formats for resource data.

   - Standard methods for querying the database.

   - Standard methods for name servers to refresh local data from
     foreign name servers.

2.4. Elements of the DNS

The DNS has three major components:

   - The DOMAIN NAME SPACE and RESOURCE RECORDS, which are
     specifications for a tree structured name space and data
     associated with the names. Conceptually, each node and leaf
     of the domain name space tree names a set of information, and
     query operations are attempts to extract specific types of
     information from a particular set. A query names the domain
     name of interest and describes the type of resource
     information that is desired. For example, the Internet
     uses some of its domain names to identify hosts; queries for
     address resources return Internet host addresses.

   - NAME SERVERS are server programs which hold information about
     the domain tree's structure and set information. A name
     server may cache structure or set information about any part
     of the domain tree, but in general a particular name server
     has complete information about a subset of the domain space,
     and pointers to other name servers that can be used to lead to
     information from any part of the domain tree. Name servers
     know the parts of the domain tree for which they have complete
     information; a name server is said to be an AUTHORITY for
     these parts of the name space. Authoritative information is
     organized into units called ZONEs, and these zones can be
     automatically distributed to the name servers which provide
     redundant service for the data in a zone.

   - RESOLVERS are programs that extract information from name
     servers in response to client requests. Resolvers must be
     able to access at least one name server and use that name
     server's information to answer a query directly, or pursue the
     query using referrals to other name servers. A resolver will
     typically be a system routine that is directly accessible to
     user programs; hence no protocol is necessary between the
     resolver and the user program.

These three components roughly correspond to the three layers or views
of the domain system:

The domain space consists of a single tree and the user can
request information from any section of the tree.

   - From the resolver's point of view, the domain system is
     composed of an unknown number of name servers. Each name

     server has one or more pieces of the whole domain tree's data,
     but the resolver views each of these databases as essentially
     static.

   - From a name server's point of view, the domain system consists
     of separate sets of local information called zones. The name
     server has local copies of some of the zones. The name server
     must periodically refresh its zones from master copies in
     local files or foreign name servers. The name server must
     concurrently process queries that arrive from resolvers.

In the interests of performance, implementations may couple these
functions. For example, a resolver on the same machine as a name server
might share a database consisting of the the zones managed by the name
server and the cache managed by the resolver.

3. DOMAIN NAME SPACE and RESOURCE RECORDS

3.1. Name space specifications and terminology

The domain name space is a tree structure. Each node and leaf on the
tree corresponds to a resource set (which may be empty). The domain
system makes no distinctions between the uses of the interior nodes and
leaves, and this memo uses the term "node" to refer to both.

Each node has a label, which is zero to 63 octets in length. Brother
nodes may not have the same label, although the same label can be used
for nodes which are not brothers. One label is reserved, and that is
the null (i.e., zero length) label used for the root.

The domain name of a node is the list of the labels on the path from the
node to the root of the tree. By convention, the labels that compose a
domain name are printed or read left to right, from the most specific
(lowest, farthest from the root) to the least specific (highest, closest
to the root).

Internally, programs that manipulate domain names should represent them
as sequences of labels, where each label is a length octet followed by
an octet string. Because all domain names end at the root, which has a
null string for a label, these internal representations can use a length
byte of zero to terminate a domain name.

By convention, domain names can be stored with arbitrary case, but
domain name comparisons for all present domain functions are done in a
case-insensitive manner, assuming an ASCII character set, and a high
order zero bit. This means that you are free to create a node with
label "A" or a node with label "a", but not both as brothers; you could
refer to either using "a" or "A". When you receive a domain name or

label, you should preserve its case. The rationale for this choice is
that we may someday need to add full binary domain names for new
services; existing services would not be changed.

When a user needs to type a domain name, the length of each label is
omitted and the labels are separated by dots ("."). Since a complete
domain name ends with the root label, this leads to a printed form which
ends in a dot. We use this property to distinguish between:

   - a character string which represents a complete domain name

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

fastcase®
Smarter legal research.