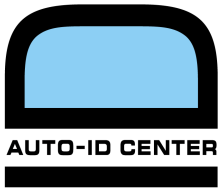
Published June 1, 2002. Distribution restricted to Sponsors until September 1, 2002.



TECHNICAL MEMO

On the design of Globally Unique Identification Schemes

Daniel W. Engels

AUTO-ID CENTER MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 77 MASSACHUSETTS AVENUE, BLDG 3-449, CAMBRIDGE, MA 02139-4307, USA

ABSTRACT

Globally Unique IDentification schemes (GUIDes) are essential for large-scale applications involving automation with networked devices. The values of the identifiers are governed by general requirements and constraints on globally unique identification schemes. In addition, application specific requirements, such as how the identifiers are to be used and allocated, may be placed upon the identifiers. We identify the requirements for a Globally Unique Identification scheme. Then, we analyze several application specific identification schemes in light of these uniqueness requirements.

Published June 1, 2002. Distribution restricted to Sponsors until September 1, 2002.

TECHNICAL MEMO

On the design of Globally Unique Identification Schemes

Biography

by Daniel W. Engels
Associate Director

Daniel W. Engels received his B.S. from the University at Buffalo, his M.S. from the University of California, Berkeley, and his Ph.D. from the Massachusetts Institute of Technology all in Electrical Engineering and Computer Science. His master's thesis is in the area of computer-aided design for electronic systems, and his doctoral thesis is in the field of theoretical computer science. Dr. Engels joined the Auto-ID Center after obtaining his doctoral degree where he leads the day-to-day research activities of the Center. Dr. Engels' research interests include scheduling theory and applications, real-time system design, distributed and mobile computing, and computer-aided design for embedded systems.

TECHNICAL MEMO

On the design of Globally Unique Identification Schemes

Contents

1. INTRODUCTION

The ability to uniquely identify items, devices, and services (collectively called objects) is essential for many applications such as security access control and communications. The inability to uniquely identify every object of interest renders many of these applications impotent (e.g., security access control either allows everybody or nobody access) or at best inefficient (e.g., communications are broadcast to every object).

The essential component for unique identification is the association of a unique identifier with the object. The identifier may take many forms, such as a given name or number (e.g., a social security number) or a characteristic of the object (e.g., a fingerprint). We are concerned with the properties of unique, given identifiers in this paper. To ensure uniqueness, these identifiers must conform to some set of rules, or a scheme, governing their assignment, their structure, and even the value of the identifier itself.

A unique identification scheme must be used to generate unique identifiers for every object of interest for a particular application or set of applications. Within a single identification scheme there are several general requirements on the identifiers, most importantly that each generated identifier be unique, and there may be application specific requirements on the identification scheme as well, possibly arising from the use and assignment of the identifiers. When identifiers from multiple identification schemes are to be used on different objects within the same application or set of applications, then the identification schemes themselves must be uniquely identifiable from the identifiers themselves to enable an identifier to be interpreted using the correct identification scheme.

We identify the general requirements for a Globally Unique IDentification scheme, or GUIDe, in Section 2. These requirements include generalized requirements due to allocation and global application constraints. While these requirements target global identification schemes, they are applicable to all unique identification schemes regardless of their scope. We then investigate the application specific requirements of a GUIDe by analyzing several existing identification schemes and the application(s) for which they were developed. In Section 3 we investigate the Media Access Control (MAC) identification scheme used to identify devices on the same segment of network such as Ethernet. In Section 4 we investigate the Internet Protocol (IP) identification scheme used to identify devices connected to the same network. In Section 5 we investigate the Domain Name Service (DNS) used to provide human-readable names to devices connected to the same network. In Section 6 we investigate the Handle identification system used to uniquely identify networked resources. In Section 7 we investigate the Electronic Product Code (EPC) used to uniquely identify objects. We summarize our findings in Section 8.

2. UNIQUE IDENTIFICATION SCHEME REQUIREMENTS

The purpose or function of a GUIDe is to provide a globally unique, persistent identifier for an object that may be used efficiently by applications. The purpose of GUIDes places general requirements on their functional capabilities, on their encodings within data streams and possibly human readable communications, and on their structure. Additional requirements may be placed on the GUIDe due to how a unique identifier is used or assigned within an application.

The general functional requirements on a GUIDe follow from the use of its identifiers for unique identification within the universe. RFC 1737, "Functional Requirements for Uniform Resource Names (URNs)," provides a set of functional and encoding requirements for unique, human readable resource names used on the Internet and World Wide Web [10]. Many of these requirements are generic to all GUIDes. We use the same names and similar descriptions where possible in our lists of general functional and encoding requirements below. The general functional requirements for a GUIDe are as follows.

Global Uniqueness
A unique identifier derived from a GUIDe (referred to as a GUIDe identifier, or simply GUIDe when it is clear that an identifier is being referred to) is an identifier assigned to at most one object in the universe.

Global Scope
A specific GUIDe identifier has the same meaning everywhere. Context does not influence the meaning or interpretation of a GUIDe.

Persistence
The lifetime of a GUIDe identifier is expected to have no limits. That is, the GUIDe identifier will be globally unique forever, and may be used to identify an object well beyond the lifetime of the object it identifies.

Scalability
The GUIDe must be able to uniquely identify objects on a global scale even if the number of objects is greater than originally thought to exist. Applications that use the GUIDe must also be able to scale without being limited by the GUIDe.

Extensibility
The GUIDe must permit future extensions, in particular to enable scalability.

Independence
The identifier issuing authority has the sole responsibility to determine the conditions under which it will issue an identifier.

There are several general requirements on the encoding of the GUIDe in addition to the functional requirements above. For a GUIDe to be useful, it must be communicated between at least two parties, the object and the application. For any communication between two parties to be effective there are two essential preconditions: 1) the existence of a common symbol set, and 2) the existence of a common semantic interpretation of these symbols. Failure of the first condition implies a failure to communicate at all, and failure of the second implies that the meaning of the communication, e.g., the GUIDe identifier, is lost.

In the case of a global system, these conditions must be strengthened to that of a unique symbol set, i.e., a **Single Encoding**, with a unique semantic interpretation. The requirement of uniqueness allows any party, either object or application, to initiate communication that can be received and understood by any other party, either intelligent object or application.

The general encoding requirements for a GUIDe are as follows.

Single Encoding
There should be exactly one encoding of a GUIDe.

Simple Comparison
The encoding of a GUIDe should enable a simple, local, and deterministic comparison algorithm between two GUIDe identifiers.

Communication Ability
The encoding of a GUIDe should be capable of being communicated easily and unmodified over a communication network.

The **Global Scope** and **Single Encoding** requirements define the property of uniqueness and rule out the ability to define a GUIDe identifier within some bounded context. In a bounded context situation, once the identifier moves out of the context in which it was defined, the meaning of the GUIDe becomes lost. This implies that GUIDes that will travel through multiple contextual and semantic domains should be distinguishable, i.e., come from different unique symbol sets, to prevent incorrect interpretations and misunderstandings. The property of uniqueness allows a GUIDe to be used unambiguously in any context, allowing the GUIDe to be passed on, referred to, and reused, while still preserving its semantics. Thus, a GUIDe is required to have well documented global semantics that enable the deterministic unique interpretation of a GUIDe identifier.

In addition to these general functional, encoding, and semantic requirements, the GUIDe identifiers may be required to have a particular structure. There are no general structural requirements on a GUIDe; therefore, all structural requirements are due to application specific constraints. There are, however, general requirements pertaining to the structure of a GUIDe identifier as they relate to the applications that use them.

Applications, in addition to properly interpreting a GUIDe identifier, must also be able to allocate and use that GUIDe without being limited by the GUIDe (the **Scalability** requirement above). Applications that require globally unique identifiers are themselves global in scope. For these applications, a scheme to uniquely identify objects is only useful if the applications are able to use the unique identifiers with some minimum level of performance.

In practice, global scalability of applications is achieved through hierarchically deployed and/or distributed services and GUIDe assignment managers. These services and managers operate most efficiently when the hierarchy of the services management structure mirrors the structured hierarchy inherent in the GUIDe and distribution of services does not cross functional GUIDe hierarchical boundaries (but distribution may occur at multiple levels).

The hierarchy of applications imposes several practical constraints on the structure of the GUIDe.

Partitioned Structure
The GUIDe identifier should have distinct structure that is either fixed or determined from the identifier itself.

Hierarchical Encapsulation
The partitioned structure itself should contain a tree-like topology of interpretation precedence. A tree may be possible without encapsulation; however, either the namespace is flattened out partially or fully or there are dead-end paths requiring backtracking and random searching.

Manageable Partition Size
The number of identifiers possible within each partition must be manageable for the applications using the identifier.

Additional requirements specific to a particular application or the administration of a GUIDe may also impact the identification scheme. The following sections investigate how specific applications impact the unique identification scheme developed for those applications.

3. MEDIA ACCESS CONTROL (MAC) IDENTIFIERS

Media Access Control (MAC) identifiers, also referred to as MAC addresses, are designed to uniquely identify a physical piece of equipment, such as a network card, connected to an IEEE project 802 compliant network such as an Ethernet or a Token Ring network [8]. MAC addresses are intended to identify items of real physical equipment, parts of such equipment such as separable subsystems, or individually addressable ports. They are not intended to identify entire devices, such as a personal computer, connected to a network.

All communications over networks such as Ethernet and Token Ring use the MAC addresses of the source and destination devices to designate the sender of the communication packet and the receiver of the communication packet respectively. IEEE project 802 networks require a 48-bit MAC address (MAC-48) for all hardware devices connected to them. Therefore, all transmission protocols that communicate over networks such as Ethernet and Token Ring must associate their protocol specific address to a MAC-48 address when the destination communication device is on the same network subnet as the source communication device. Each device maintains a table mapping protocol specific addresses to MAC-48 addresses for this communication. Discovery algorithms to determine this mapping also exist.

The MAC-48 identifiers consist of two partitions. The first partition is the 24-bit IEEE Organizationally Unique Identifier, or OUI. The OUI uniquely identifiers the owner of the MAC address space beginning with the OUI. The second partition is the 24-bit number assigned by the owner of the OUI for that MAC address. The 24-bit number is assigned according to ANSI/IEEE Std 802 to generate unique MAC addresses. No provision is made for the extensibility of a MAC-48 identifier.

The Institute of Electrical and Electronics Engineers, Inc. (IEEE) has been designated by the ISO Council to act as the registration authority for the implementation of International Standards in the ISO/IEC 8802 series which include the IEEE project 802 standards. The IEEE is the one world-wide source of registered OUIs.

The structure of a MAC address between a 24-bit OUI and a 24-bit number provides a hierarchically encapsulated addressing scheme. The encapsulated hierarchy enables the allocation of MAC addresses to be performed in a manageable, distributed manner.

MAC addresses are used as a flat namespace for communication purposes since only devices connected to the same segment of an IEEE project 802 network communicate with one another using MAC addresses. Devices on separate network segments communicate using higher-level communication protocols through network connection devices such as routers and gateways.

4. INTERNET PROTOCOL (IP)

The Internet is a global network of computers created through connected local networks that communicate through devices such as routers and gateways. The Internet technically requires the existence of a globally unique public namespace, i.e., globally unique identifiers, to remain a global network. The Internet Protocol (IP) address is a GUIDe designed to provide this globally unique public namespace. An IP address uniquely identifies network interfaces connected to a network generally and the Internet particularly [5].

IP addresses were designed to be used with the Internet Protocol (IP) packet routing service, a higher level communication protocol designed to enable systems on different connected networks to communicate. One of the primary application specific design requirements for IP addresses was to enable routers to quickly distinguish between different logical networks. This requirement caused the IP designers to structure the IP address in a hierarchically encapsulated manner that enables the routers to quickly identify network addresses. The IP address hierarchy provides a logical hierarchy rather than a geographical hierarchy. The IP packet routing service uses this logical hierarchy to send communication packets from one network interface to another network interface over the network. Routers need only to get an IP communication packet to the correct network. Therefore, they need only access the correct partition within the hierarchy to determine the correct sub-network to forward the communication packet on to. The hierarchically encapsulated structure also enables a router to store information about the sibling networks within their branch of the IP address space only.

The most widely used version of IP addressing is IP Version 4 (IPv4). An IPv4 address is a 32-bit identifier. The first **n** bits of an IP address are used to identify the class of the IP address. The value of **n** ranges from 1 to 4, depending upon the class of the address. The next **m** bits of an IP address are used to identify the individual network. The value of **m** depends upon the class of address. The remaining 32 – **n** – **m** bits are used to uniquely identify the local host within the network. The five classes of IPv4 addresses are given in Table 1.

Table 1: The class structure for IPv4 addresses.

CLASS	CLASS IDENTIFIER	NETWORK IDENTIFIER LENGTH (BITS)	LOCAL HOST IDENTIFIER LENGHT (BITS)
A	0	7	24
B	10	14	16
C	110	21	8
D	1110	28-bit multicast group address	
E	1111	reserved for experiments	

The notion of address classes has become less important with the development of classless addressing that provides for variable length network identifiers [9] [3]. Additional hierarchy within the networks is provided by further subdividing the local identifier into a variable length subnet identifier followed by a host identifier (with the total length of the subnet identifier and host identifier being 32 – **n** – **m** bits). The length of the subnet identifier must be specified separately using a subnet mask. The subnet identifier enables a network to consist of a collection of disjoint networks. The IPv4 addressing scheme does not provide for extensibility. This deficiency is addressed in the latest version of IP, IP Version 6 (IPv6) [4], that has seen only limited use to date.

The Internet Assigned Numbers Authority (IANA) is the registration authority for the assignment of unique network identifiers for IP addresses. The IANA is chartered by the Internet Society (ISOC) and the Federal Network Council (FNC) to act as the clearinghouse to assign and coordinate the use of numerous Internet protocol parameters including IP network identifiers. The owner of a unique network identifier is responsible for the allocation and assignment of subnet identifiers and host identifiers under the namespace defined by the given unique network identifier.

The structure of an IP address provides a hierarchically encapsulated addressing scheme. The encapsulated hierarchy enables the allocation of IP addresses to be performed in a manageable, distributed manner.

The topology of assigned IP addresses logically forms a tree with internal nodes in the tree identifying network identifiers and subnets, while the leaves identify the network interface host identifiers. Routers rely upon this logical structure to efficiently route communication packets to the correct network through sibling routers, that is routers at the same level and in the same branch of the IP address topology. A routing table is used by the routers to indicate the fastest path towards the destination IP address. The routing table is updated periodically to reflect the current status of the network. Applications use IP addresses to communicate between devices located in different networks.

5. DOMAIN NAME SYSTEM

The Domain Name System is used to uniquely name resources connected to a network [6] [7]. The Domain Name System provides a consistent name space independent of resource specific and protocol specific information. The Domain Name Space is a tree structured namespace where each node in the tree contains a label that is zero to 63 octets in length. Labels must start with a letter and can only consist of letters, digits and hyphens. Sibling nodes may not have identical labels, but labels may be identical between non-sibling nodes. The zero length null label is reserved for the root node of the tree.

The domain name of a node is the 'dot' separated list of the labels on the path from the node to the root of the tree. By convention, the labels are appended as one moves towards the root node. Thus, the label for a node C is the left-most label in the domain name for that node C.B.A, where node B is the parent to node C and node A is the parent to node B (and the root node is the parent to node A) in the tree structure.

The primary service that uses the domain names is the Domain Name Service (DNS). DNS is a directory service that maps a domain name to an IP address (or set of IP addresses) assigned to that domain name. DNS is designed "to provide a mechanism for naming resources in such a way that the names are mappable into IP addresses and are usable in different hosts, networks, protocol families, internets, and administrative organizations." [7].

Internet domain name translation was originally performed by searching local copies of a table of all domain names. This table became too large to update and distribute in a timely manner and too large to fit into many resources, so the DNS was invented.

The DNS creates a distributed database used primarily for the translation between domain names and IP addresses. The DNS consists of two logically distinct parts: name servers and resolvers.

Domain name servers store authoritative data about certain sections of the database and answer queries about the data. Domain name servers typically have authoritative data about a specific branch, or zone, of the domain name tree structure. Domain resolvers query domain name servers for data on behalf of user processes. Every resource therefore needs a DNS resolver; some resources will also need to run

domain name servers. Since no name server has complete information, in general it is necessary to obtain information from more than one name server to resolve a query.

The DNS administration model is for DNS names to be managed by the network administrator(s) at the domain name zone level, with no provision for a per name administrative structure, and no facilities for anyone other than network administrators to create or manage names. That is, DNS is designed to have a single name administrator at each node in the domain naming hierarchy. The Internet Corporation for Assigned Names and Numbers (ICANN) is the organization that administers the top-level domain name servers. ICANN is the global, non-profit coordinating body that ensures that the DNS continues to function effectively by overseeing the distribution of unique numeric IP addresses and domain names. Among its other responsibilities, ICANN oversees the processes and systems that ensure that each domain name maps to the correct IP address. A select set of authorized companies may be used to register domain names with ICANN.

The structure of a domain name provides a hierarchically encapsulated naming scheme. The encapsulated hierarchy enables the allocation of domain names to be performed in a manageable and distributed manner.

The Domain Name Service relies upon the domain name hierarchy to provide efficient mappings between domain names and IP addresses. Each internal node in the domain name tree structure requires that the owner of that node provide a domain name server that identifies its children nodes.

6. HANDLE SYSTEM

The Handle System is a distributed information system designed to provide an efficient, extensible, and secure global name service. The Handle System provides a general purpose global name service designed to manage unique identifiers for digital objects while allowing secured name resolution and administration over the Internet [11]. The Handle System is essentially a generalization of the Domain Name System. The Handle namespace is designed to uniquely identify all digital objects (not just systems as in the Domain Name System) with the Handle resolver providing more resolving services than DNS.

The Handle protocol enables a distributed computer system to store identifiers of digital resources and resolve those identifiers into the information necessary to locate, access, and otherwise make use of resources. Associated values may be modified to reflect the changing state of the resource; thus, the identifier may remain unchanged over changes of location and other current state changes. Each identifier may have its own administrator, and administration may be performed in a distributed environment. The identifier to value bindings may be secured, allowing identifiers to be used in trust management applications.

The Handle namespace is a federated namespace that allows any existing local namespace to join the global Handle namespace by obtaining a unique Handle System naming authority. Local names and their value binding(s) remain intact after joining the Handle System. A local name, when combined with its unique naming authority, is guaranteed to be unique under the global Handle namespace.

The Handle System has been designed to serve as a naming system for very large numbers of entities and to allow administration at the name level (as opposed to DNS which allows administration at the domain zone level). The Handle System data model allows access control to be defined at the level of each value. Each identifier can further define its own administrator(s) to manage the handle data via the Handle System authentication protocol.

The Handle namespace may be considered as a superset of many local namespaces, with each local namespace having its own unique Handle naming authority. Just as with domain names, the Handle namespace forms a tree structure with each node containing a label, or Handle. A unique identifier is created within the global Handle namespace by concatenating the globally unique Handle naming authority with a unique local name under the naming authority. Thus, every identifier consists of two parts: its Handle naming authority (prefix) and a unique local name under the naming authority (suffix).

```
<identifier> ::= <Handle naming authority>"/"<Handle local identifier>
```

Handle identifiers may consist of any printable characters from the Universal Character Set, two octet form (UCS-2) of ISO/IEC 10646, which is the exact character set defined by Unicode v3.0. Unlike DNS, Handle naming authorities are defined in a hierarchical, i.e., tree, fashion. That is, the parent nodes in the naming authority tree hierarchy form the prefix to the unique identifiers. The Corporation for National Research Initiatives (CNRI) developed the Handle System and administers the allocation of Handle naming authorities. Users of Handle System include the Library of Congress, the Defense Technical Information Center (DTIC), and the International DOI Foundation (IDF).

The structure of a Handle identifier provides a hierarchically encapsulated identification scheme. The encapsulated hierarchy enables the allocation of Handle identifiers to be performed in a manageable, distributed manner.

Handle resolvers rely upon the encapsulated hierarchy of Handle identifiers in order to obtain information about the identified resource, such as its IP address. Each node in the Handle namespace tree structure requires that the owner of that node either provides a Handle resolver or identifies a Handle resolver that maintains information about it.

7. OBJECT NAME SYSTEM

The Object Name System is designed to uniquely name physical objects and use those unique names as pointers to information about the associated object located somewhere over the network. The primary identification scheme used within the Object Name System is the Electronic Product Code. The Electronic Product Code (EPC) is an identification scheme designed to uniquely identify physical objects [2] [1]. The EPC provides a consistent namespace independent of object specific information.

The Electronic Product Code namespace is segmented into four hierarchically encapsulated partitions: version, domain manager, object class, and serial number. This segmentation gives the EPC namespace a singly rooted out-tree topology with a height of exactly four: depth 0 – root node, depth 1 – version number, depth 2 – domain manager, depth 3 – object class, and depth 4 – serial number. Each node in the namespace topology is labeled with a zero padded binary number (the total number of bits in this number is determined by the version number parent node). Sibling nodes may not have identical labels, but labels may be identical between non-sibling nodes. The null label is reserved for the root node of the tree.

The paths from the root node to the leaf nodes of the tree correspond to valid EPC identifiers. The value of a path's EPC identifier is determined by concatenating the labels of the nodes on the path from the root node to the leaf node. EPC identifiers that differ only in their version number, that is their domain manager, object class, and serial numbers are identical (ignoring leading zeroes), are defined to be identical; thus, they correspond to the same physical object.

The primary service that uses the EPC is the Object Name Service (ONS). ONS is a directory service built upon DNS. It maps an EPC to an IP address (or set of IP addresses) assigned to resources that contain information about the object with the given EPC. ONS creates a distributed, multi-rooted database used primarily for the translation between EPCs and IP addresses.

The Auto-ID Center manages the permissible version numbers in valid EPCs. The version number indicates the total number of bits in an EPC as well as the number of bits within each partition of the EPC. There are currently four version numbers defined with all additional version numbers reserved for future use. The Auto-ID Center also manages the allocation of the domain manager numbers. The Auto-ID Center is the one world source for domain manager numbers. The owner of a domain manager number manages the allocation of class codes, and similarly, the owner of a class code manages the allocation of serial numbers. The tree structure of the EPC namespace captures this allocation management structure.

The hierarchically encapsulated EPC namespace structure enables the allocation of EPC identifiers to be performed in a manageable, distributed manner. The namespace structure also enables similar physical objects, such as all Gillette Mach 3 four packs, to have identifiers that differ only in their serial number. Similar identifiers enable the compaction of identical information about similar objects; thereby reducing storage requirements and minimizing the size of the ONS resolver tables.

The Object Name Service relies upon the encapsulated hierarchy EPC structure to provide efficient mappings between EPCs and IP addresses.

8. SUMMARY

A globally unique identifier (a GUIDe identifier) must be an unambiguously unique identifier specified by its single encoding and its context free semantics. There are general requirements placed upon a GUIDe including its uniqueness, scope, extensibility, encoding, semantics, and structure. These general requirements must be satisfied within the context of a particular application or set of applications. A GUIDe identifier's specific encoding, semantics, and structure are largely determined by the application or set of applications for which it is designed. This application specific nature limits a GUIDe's general use within applications for which it was not designed.

The allocation and management of a GUIDe's namespace is at least as important to the structure of the GUIDe as are the applications that use the GUIDe identifiers. For example, the allocation and management of MAC addresses forces its hierarchical structure (This is true for all of the identification schemes we have examined here.). However, the applications that use MAC addresses do not rely upon this hierarchy for their functionality. In contrast, the communication/routing applications that use IP addresses rely upon the structured nature of the address (although not the structure itself) to work efficiently; while the Domain Name Service relies upon the explicit structure of domain names to function correctly.

Globally unique identifiers may form the basis upon which new globally unique identifiers are created. The Handle System illustrates how a globally unique identifier may be obtained by the concatenation of a globally unique identifier with a locally unique identifier. The globally unique identifier and the locally unique identifier may contain different encodings, semantics, and structure; however, how the two identifiers are concatenated must be defined globally. The concatenated identifiers are separated by a forward slash '/' within the Handle System.

9. REFERENCES

1. **David Brock. The Compact Electronic Product Code – A 64-bit Representation of the Electronic Product Code.**
   Technical Report MIT-AUTOID-WH-008, MIT Auto-ID Center, Massachusetts Institute of Technology, November 2001.

2. **David Brock. The Electronic Product Code – A Naming Scheme for Physical Objects.**
   Technical Report MIT-AUTOID-WH-002, MIT Auto-ID Center, Massachusetts Institute of Technology, January 2001.

3. **V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy.**
   IETF, RFC 1519, September 1993. http://www.ietf.org/rfc/rfc1519.txt.

4. **R. Hinden and S. Deering. IP Version 6 Addressing Architecture.**
   IETF, RFC 2373, July 1998. http://www.ietf.org/rfc/rfc2373.txt.

5. **Internet Protocol: DARPA Internet Program Protocol Specification.**
   University of Southern California Information Sciences Institute.
   IETF, RFC 791, September 1981. http://www.ietf.org/rfc/rfc0791.txt.

6. **P. Mockapetris. Domain Names – Concepts and Facilities.**
   IETF, RFC 1034, November 1987. http://www.ietf.org/rfc/rfc1034.txt.

7. **P. Mockapetris. Domain Names – Implementation and Specification.**
   IETF, RFC 1035, November 1987. http://www.ietf.org/rfc/rfc1035.txt.

8. **David C. Plummer. An Ethernet Address Resolution Protocol.**
   IETF, RFC 826, November 1982. http://www.ietf.org/rfc/rfc0826.txt.

9. **Y. Rekhter and T. Li. An Architecture for IP Address Allocation with CIDR.**
   IETF, RFC 1518, September 1993.
   http://www.ietf.org/rfc/rfc1518.txt.

10. **K. Sollins and L. Masinter. Functional Requirements for Uniform Resource Names.**
    IETF, RFC 1737, December 1994.
    http://www.ietf.org/rfc/rfc1737.txt.

11. **Sam Sun and Larry Lannom. Handle System Overview.**
    IETF, Internet Draft: Handle System Overview, February 2002.
    http://www.ietf.org/internet-drafts/draft-sun-handle-system-08.txt.

Designed and produced by Foxner. www.foxner.com