Design of the Common Directory Interface for DECnet/OSI

Richard L. Rosenbaum
Stanley I. Goldfarb

Digital has developed the Common Directory Interface (CDI) as the means by which DECnet/OSI can now access and manage node name and address information in multiple directory services. CDI comprises libraries for node name-to-address translation and a tool set for managing and migrating node information among different directory services. The Common Directory Registration API is layered on top of a set of directory service wrapper routines to provide an extensible mechanism for adding new directory services. CDI gives customers greater flexibility in choosing a directory service and supports the new multiprotocol capabilities in DECnet/OSI, which support the open systems interconnection (OSI) standards.

The Common Directory Interface (CDI) provides the ability to store and retrieve DECnet node information from a variety of directory services. It consists of the CDI library, which enables multiple directory access, and the CDI registration tool set, which creates and maintains node/addressing information in multiple directory services. CDI was developed for the DECnet/OSI for OpenVMS operating system version 6.0 and for the DECnet/OSI for Digital UNIX operating system version 3.0.

This paper begins by presenting the product goals and the background of the CDI design. It then discusses the structure of the CDI components, the CDI library, and the CDI registration tool set.

Design Goals

As the interface to DECnet node information from multiple directory services, CDI was designed to meet the following goals:

- Give DECnet network administrators and users a choice of directory services.
- Provide system administrators with an easy-to-use node registration tool.
- Enable easy and flexible configuration of directory choices.
- Provide developers of the DECnet protocol software with a simple internal interface that hides the complexities and differences between the various directory services.
- Provide a common design for both DECnet/OSI platforms: the OpenVMS and the Digital UNIX operating systems.
- Interoperate with older, non-CDI systems.

Background

In 1991, Digital updated its DECnet networking products to include the use of the DECdns distributed directory service.[1] DECdns provided a highly scalable, distributed information source for translating node names to addresses and addresses to node names.

Initially, customer acceptance of this name service was low for a number of reasons:

- Adoption of this new technology required a significant learning curve.
- Significant planning was required before the DECdns service could be deployed.
- Users of small networks did not need the features of a distributed naming service—the costs outweighed the benefits. These customers requested a naming service based on local files similar to the Phase IV DECnet product.
- Customers were deploying a number of other directory services—in particular the Domain Name System—for storing host information for transmission control protocol/internet protocol (TCP/IP) networks.[2]
- A new comprehensive service, X.500, had the advantage of being an international standard.[3]

These reasons, together with the need to directly support TCP/IP host names and addresses, prompted Digital to incorporate new directory service choices in a new release of DECnet/OSI software.

CDI: Basic Design

Supporting multiple name services required decisions to be made concerning naming syntax, multiple address formats, and local file support. These decisions affected the design of both the CDI library and the CDI registration tool set.

Client-based versus Server-based Design

The earliest and most fundamental design decision was choosing between a client-based or a server-based solution. With a client-based design, support for the various directory services would be accomplished through a variety of client-based programming libraries. With a server-based design, a single client library would communicate with a new "multiheaded" server that would fan out to the directory servers.

Since clients outnumber servers, a client-based approach affects more systems during the upgrade process. In spite of this drawback, we chose a client-based solution for the following reasons:

- Implementation of the client-based design would be less complex than the server design.
- A client-based design did not have the syntax and protocol translation issues of a server-based design.
- With a server-based solution, client changes would still be required to support new native naming syntaxes.
- For small installations, no server would be needed if node information was stored in a local file: local file support was not possible with a server-only approach.

Naming Syntax

One of the most visible complications when supporting multiple naming services is the need to recognize different name syntaxes. Table 1 gives the different syntaxes for three widely used directory services.

A further complication of supporting different name syntaxes was the use of an internal DECdns name format by network management. One of the goals of the CDI design was to allow management requests to be exchanged with older, non-CDI systems.

For the initial implementation, CDI continues to support the internal DECdns format, rather than use a newer, non-DECdns specific format alongside the existing one. As a result, CDI is required to map non-DECdns names onto the DECdns format. For example, the name *hq.xyz.com* from the Domain Name System maps onto the DECdns name *DOMAIN:.hq.xyz.com* (actually onto the internal DECdns form of this name).

Multiple Address Support

Along with the introduction of CDI, a major innovation in this release of DECnet/OSI was direct support for TCP/IP transports in addition to the existing

Table 1
Naming Syntax

Directory Service	Example Name
DECdns	XYZ:.hq.sales.system1
Domain Name System	system1.sales.hq.xyz.com
X.500	/c=US/O=XYZ/ou=hq/ou=sales/ap=system1/ae=DECnet

Notes:
The X.500 service is not supported by the first release of CDI.
The syntax shown for X.500 is commonly used but is not part of a standard.

support for DECnet Phase IV and OSI. To simplify the initial implementation, IP addresses are retrieved only from the Domain Name System (not from DECdns). However, the design of CDI allows the retrieval of both kinds of addresses from any supported directory; for example, OSI addresses can be obtained from the Domain Name System.[4,5]

Support of multiple protocols created another naming issue. Many customers already have a Domain Name System in place in their networks. Often DECnet systems are also running TCP/IP protocols and are registered in the Domain Name System, yet these systems are not running DECnet software over TCP/IP. For example, a system registered as hq.xyz.com may be directly reachable with TCP/IP but not with DECnet over TCP/IP. In this case, it is possible that CDI may retrieve a valid IP address for a remote system that is unreachable by the DECnet protocol.

For these reasons, when CDI determines that both the Domain Name System and the DECdns naming service (or a local file) are specified in the search path, it does not stop processing the search path until both the IP address and the OSI address have been obtained, or until the end of the list has been reached. In this way, if the desired remote system is not running DECnet over TCP/IP, an attempt to connect will be made through the DECnet protocol, using a connectionless network service (CLNS) OSI address.

Local File Support

Early versions of the DECnet networking product offered only a local file for node-to-address information. The first release of DECnet/OSI replaced the local file with the DECdns naming service. Unfortunately, administrators of small- and medium-sized networks found that the benefits of DECdns (scaling and centralized management) were outweighed by its additional complexity.

A subsequent version of DECnet/OSI introduced the Local Naming Option. This allowed approximately 150 nodes to be stored in a local file, but many customers found this number to be too small.

CDI supports a very large local file: the supported limit is 100,000 nodes, but there is no fixed internal limit. In addition, through the use of the search path, customers can configure the local file either as a backup to a distributed service, or as a way to provide greater performance. Note that both of these qualities are also provided in a more automated way by the CDI cache (see the CDI Library Cache section for more information).

Security Considerations

CDI relies upon the security provided by the underlying directory services (or in the case of the local file, the file system). Security of its remote management features depends on the network management security system.

CDI Libraries: Basic Design

CDI is implemented as shared libraries on both the Digital UNIX and the OpenVMS operating systems. At the highest level, the design is identical on both systems, as shown in Figure 1. Name-to-address translation requests from the session control layer are passed through a single entry point in each CDI library.

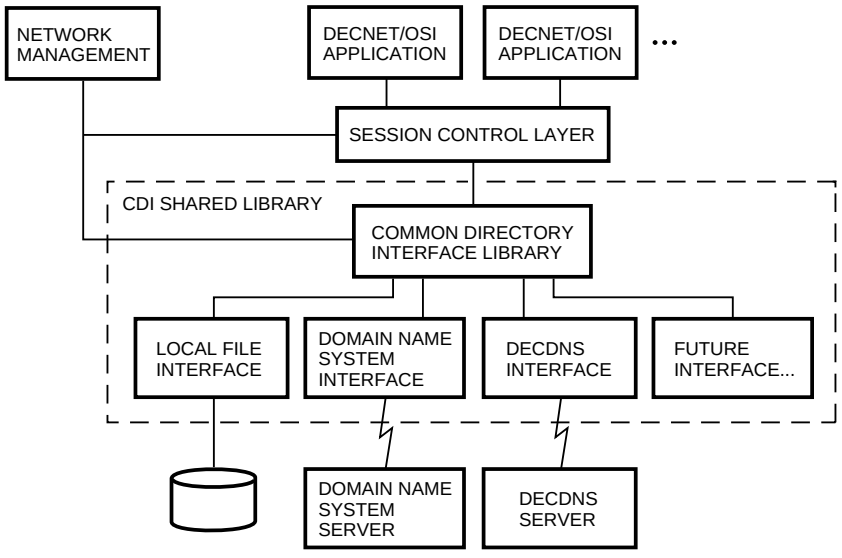


Figure 1
Block Diagram of the CDI Library

Depending upon the search path (described below), the CDI libraries translate and forward the request to one or more directory services (or they look up the information in a local file).

The CDI implementation was considerably more complex on the OpenVMS operating system than on the Digital UNIX operating system due to the differing design of DECnet/OSI on each system. On the Digital UNIX operating system, the DECnet/OSI session control layer consists of a shared library that is linked with each network application. Name resolution requests are processed synchronously. On the OpenVMS operating system, session control is a component of the NET$ACP process. Since all name resolution requests are channeled through this single process, operations must be asynchronous (requests must block concurrent operations). In addition, since multiple requests may be simultaneously outstanding, the library is multithreaded. Asynchronous, multithreaded operations on the OpenVMS operating system are implemented using the asynchronous system trap (AST) mechanism. For these reasons, the CDI implementation on OpenVMS was much larger and more complex.

CDI Search Path

Another goal was to permit flexibility in determining a configuration of directory services. The CDI design achieves this goal in two ways. First, it allows administrators to select their service(s) of choice and to use them in any order. The search path is normally created during network configuration and can be subsequently managed either locally or remotely. Second, it gives network users the ability to use short, abbreviated names instead of potentially cumbersome full names. For example, they can use "system1" instead of "system1.sales.hq.xyz.com."

A single mechanism in the CDI library—the CDI search path—provides these two capabilities. The search path consists of a series of directory service/name template pairs, as shown in Figure 2a. When the CDI library is given a name to process, it scans the search path, replacing the "*" in the name template with the supplied name. For example, if the library was searching for the name *frodo,* it would use the directory services identified from the names generated shown in Figure 2b.

During network configuration, a default search path is automatically configured based upon the local node name and the administrator-specified directory services. This search path behavior is similar to a number of existing TCP/IP host name/address lookup implementations.

CDI Library Cache

Occasionally, name service lookups can take a long time to complete (for example, if requests are traversing a slow network link, a lookup could take several seconds). To improve performance, the CDI library incorporates a single cache that accumulates node information from all the directory services. Usually, the cache is consulted before sending a request to a remote service. However, if session control determines that cached information is stale—for example, if connection to a node at a cached address reaches a node with a different name—it will reissue the call, requesting that the cache be bypassed.

Each entry in the cache has a creation time stored with it. The cache itself has a "time-to-live" value that can be modified by the administrator. If a cache lookup finds an entry whose lifetime (time since it was created) is greater than the time-to-live value, the cache entry is purged.

To prevent a period of low performance immediately after system start-up, the cache is preserved across system reboots by periodically checkpointing it to a disk file. The checkpoint interval is adjustable by the administrator.

DECdns	*
DECdns	XYZ:.hq.sales.*
DECdns	XYZ:.DNA_Node_synonym.*
Domain	*
Domain	*.sales.hq.xyz.com

(a) Directory Service/Name Template Pairs

frodo	(DECdns)
XYZ:.hq.sales.frodo	(DECdns)
XYZ:.DNA_Node_synonym.frodo	(DECdns)
frodo	(Domain)
frodo.sales.hq.xyz.com	(Domain)

(b) Address Lookup for Name *frodo*

Figure 2
Using the CDI Search Path

CDI Registration Tool: Basic Design

The CDI registration tool provides functions to create, modify, rename, display, and delete node name and address information in any of the supported directory services. It runs on the major DECnet/OSI platforms, the OpenVMS and the Digital UNIX operating systems.

The basic requirements for the CDI registration tool were the same as those for the CDI library. These three requirements were the need to:

- Support different directory services for storing node information
- Access each directory service using the appropriate application programming interfaces (APIs)

- Store data in each directory service using the appropriate data types

In addition, the following requirements were specific to the CDI registration tool:

- Both a forms and console user interface had to be provided. These had to work identically on all DECnet/OSI operating system platforms.
- Functions to transfer node information between the various directory services had to be provided.
- Other applications such as the DECnet/OSI network control language (NCL) utility and other namespace management tools had to be able to access node name management functions.

The directory services supported by the CDI registration tool are slightly different from those supported by the CDI library. The CDI registration tool supports the DECdns, the local file, and the DECnet Phase IV database services.

The DECnet Phase IV database is supported by the CDI registration tool to allow administrators to use old Phase IV node information when populating the node names and addresses for DECnet/OSI. The Phase IV database is not supported for node name-to-address lookup by the CDI library.

Due to its lack of a remote update capability, the Domain Name System is not supported by the CDI registration tool. Node name-to-address information in the Domain Name System is managed using its native tools. Dynamic updating of the Domain Name System servers is currently under study by the Internet Engineering Task Force (IETF) Domain Name System Working Group.

Application Design

The design of the CDI registration tool uses a client-based, multilayer approach. It is layered on top of a specialized API, called the Common Directory Registration (CDR) API. The CDR API differs from the API provided by the CDI library in that it presents a full set of management operations, rather than just the lookup operations required by DECnet/OSI.

In this design, the CDI registration tool provides forms and console user interfaces for node information management. It also provides functions beyond the basic ones provided by the CDR API, such as exporting from and importing to a directory service. The function of the CDR API is to perform all underlying node name management operations in a standardized manner. This layered approach was adopted to make node name management functions available to applications other than the CDI registration tool.

The CDR API defines a node definition object. This contains all the information that is exchanged between the CDR API and the application and is a canonical, directory-service-independent data representation of all information needed by the CDR API to manage node names and addresses.

To provide an extensible mechanism for adding new directory services, the CDR API is layered on top of a set of directory service wrapper routines, one per supported directory service. Access to these wrapper routines is provided by a set of entry point tables that can be extended to support new directory services. The CDR API is responsible for accepting application requests and dispatching them to the correct directory service by means of the appropriate wrapper routine. The CDR API wrapper routines are described later in this section.

Figure 3 shows the design of the CDI registration tool and the CDR API.

CDI Registration Tool User Interface

The forms and the console user interfaces had to present exactly the same characteristics on both the OpenVMS and the Digital UNIX operating systems. Because no high-level software packages at the time could provide this level of user interface portability, we developed them for this application.

The console user interface parses commands and dispatches them to the appropriate user request processing routine, using a portable command parser.

The forms user interface obtains input from task-specific forms and dispatches the function or functions associated with the form to the appropriate user request processing routine. The forms processor was written specifically for this application because no existing libraries could provide the required level of portability.

CDI Registration Tool User Request Processing

Each user request maps into a specific request processing function as follows:

- Register. Create a new node name entry in the directory service.
- Add address. Add address information to a node name entry.
- Remove address. Remove address information from a node name entry.
- Modify address. Replace the address information in a node name entry.
- Update address. Replace the address information in one or more node name entries, using information obtained from the nodes themselves (if possible).
- Modify synonym. Replace the node synonym in a node name entry.
- Rename. Change the name of a node name entry.
- Show. Display the information contained in one or more node name entries.

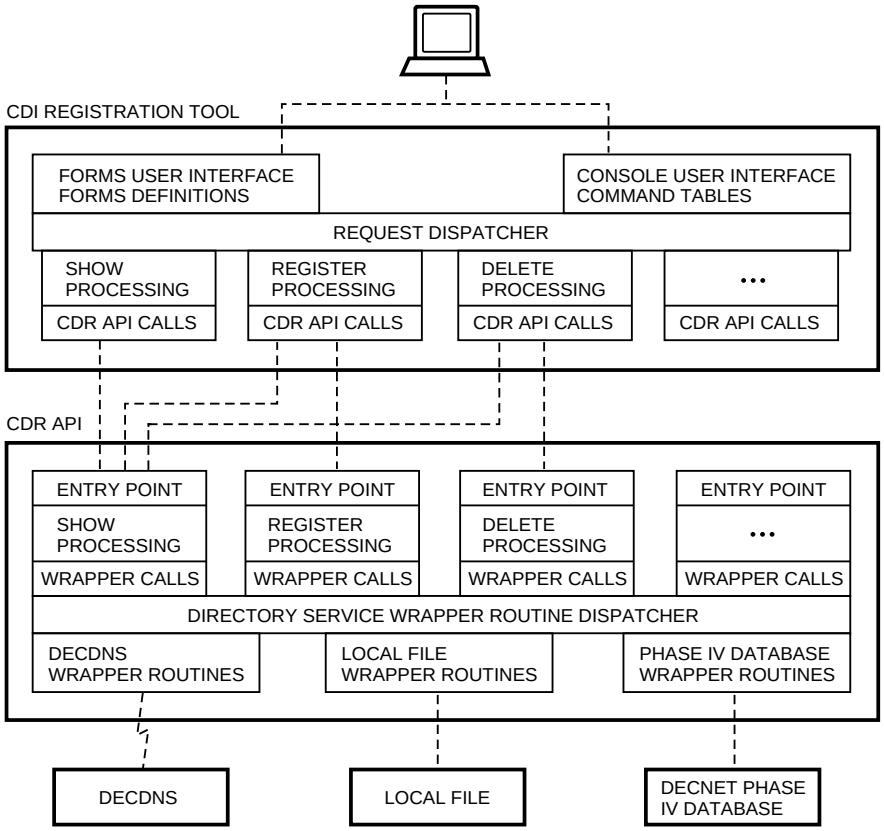


Figure 3
Block Diagram of the CDI Registration Tool and the CDR API

- Deregister. Delete one or more node name entries by name, synonym, or address.
- Repair. Fix any detected problems or inconsistencies in the directory service for one or more node name entries.
- Export. Copy the information for one or more node name entries from the directory service into a text file that can be copied between systems, edited if necessary, and imported into any other directory service.
- Import. Use an export text file to register, modify, or deregister node name entries in a directory service.

The request processing routines perform any required validation of the user request and translate those requests to calls into the CDR API. Each request may map into one or more CDR API calls, depending on the complexity of the request. For example, register and deregister requests both map into single CDR API calls, and export and import requests map into several calls.

Most requests are straightforward in their processing requirements. For example, a register request simply calls the CDR API register entry point. The CDR API takes care of any complications in processing the request.

Some requests can operate over multiple node name entries. For example, the show request enumerates the node name entries, retrieves the information contained in each node name entry, and displays the information to the user.

An export request is similar to a show request, except that the resulting information is written to a text file in a standard format instead of being displayed to the user. The import request, however, is more complicated. This request must enumerate and show the contents of the directory service, and then compare the results with the contents of the text file. Based on the specific form of the import request, it may then register new node name entries, update the information in existing node name entries, or deregister listed node name entries.

The export and import requests make use of a text file to provide maximum flexibility. The use of a text file allows the information to be copied between dissimilar platforms such as the OpenVMS and the Digital UNIX operating systems, and allows the information to be manipulated using standard tools such as batch files, grep, awk, and text editors. This is particularly useful when applying a change to all node entries.

For example, the contents of a directory service could be exported to a text file, the addresses in the text file changed to reflect a new routing area, and the results imported back into the directory to update the existing information.

The repair function performs a show operation on all specified node names to determine if any consistency errors are found. This type of error can occur in directory services that keep multiple physical records for each logical node name entry. DECdns is one example of this kind of directory service, because it uses soft links to map node synonyms and addresses back to their respective node name entries. If this type of error is found, the repair function re-registers the node synonym and address information to correct these inconsistencies.

The most complicated request is the update request. This performs a show request for the specified node names and attempts to use the current addressing information contained in the node name entry to make a network management connection to the node itself. For each node name entry, it steps through the complete set of registered addresses and tries each address in turn, using both a DECnet Phase IV connect and a DECnet/OSI connect. If a connect attempt is successful, it uses the appropriate network management requests to read the true addressing data. It then compares this addressing data to what it found in the directory service and makes any necessary corrections to the node name entry. The update operation does not operate on IP addresses due to the lack of dynamic update capabilities in the Domain Name System servers.

Before making the CDR API calls, all request processing routines convert the user request data into a node definition object, which is discussed in the next section.

CDR API Node Definition Object

The node definition object is the only input data provided to any of the CDR API entry points. It stores the necessary data for any directory service operation, using a canonical representation. The node definition object contains the following:

1. Type of directory service to access
2. Name of the node entry to access (depending on the operation being performed, it may allow a fully qualified name, a synonym, an address, or wildcards)
3. Synonym name (for DECnet Phase IV access)
4. DECnet Phase IV network service access point (NSAP) prefix (for use when a Phase IV address is specified)
5. Address information
6. Directory names used for reverse mapping of synonym names and addresses back to the fully qualified node name

The CDR API controls all access to elements within the node definition object, which further isolates the calling application from the lower-level data structures.

CDR API Entry Points

Each CDR API entry point provides one logical function to the calling application. Each user request can translate into one or more CDR API functions. The functions are

- Register. Create a new node name entry in the directory service.
- Add address. Add address information to a node name entry.
- Remove address. Remove address information from a node name entry.
- Modify address. Replace the address information in a node name entry.
- Modify synonym. Replace the node synonym in a node name entry.
- Rename. Change the name of a node name entry.
- Show. Return the information contained in one or more node name entries.
- Deregister. Delete one or more node name entries by name, synonym, or address.
- Enumerate. Return a series of node name entries, one at a time, based on a wildcard specification.

All node information passed to and from the CDR API is in the form of a node definition object, as described previously. The CDR API functions validate the canonical information contained in the node definition object and dispatch a directory-service-specific function to handle the request.

CDR API Directory Service Wrapper Routines

Each directory service supported by the CDR API has an associated set of directory service management wrapper routines. These routines provide entry points that are functionally identical to those provided by the CDR API. The CDR API does the initial input argument validation, and the directory service wrapper routines perform the data manipulation in the underlying directory service.

The CDR API dispatches the appropriate directory service wrapper routine using a set of entry point tables. This provides a means to easily extend the CDR API to include additional directory services in future versions.

CDR API Wrapper Routines for DECdns

In the DECdns name service, each node name entry contains all the information required to translate a node name to a synonym or a set of node addresses. However, no search mechanism exists to allow a

lookup of the node name entry based on the synonym or on an address. For this reason, all functions that create, modify, and delete node name entries (register, modify addresses, modify synonym, rename, and deregister) must also create, modify, and delete reverse mapping entries.

Reverse mapping entries are based on a node's synonym and addresses; they contain pointers to the true node name entry. These entries are used by the CDI library lookup functions and by the CDR API display functions (show and enumerate) to access the node name entry when given a synonym or address.

The use of reverse mapping entries requires that multiple directory service entries be created for each registered node. These must be synchronized by properly ordering the creation and deletion of the various entries when registering, modifying, or deregistering a node name. For example, when registering, the node name entry is created and its synonym and address values are set before the reverse mapping entries are created and set. Similarly, when deregistering, the reverse mapping entries are deleted before the node name entry is deleted. This prevents orphaned reverse mapping entries from being created, because they can always be found by starting from the information contained in the node name entry.

The repair function is provided in case a register or deregister operation fails before completion. The repair function corrects the reverse mapping entries by re-registering all node name entries that show errors. The CDI registration tool (not the CDR API) provides this higher-level function.

CDR API Wrapper Routines for the Local Node File

Under the OpenVMS operating system, the local node name file is implemented using a record management system (RMS)-indexed file. Under the Digital UNIX operating system, a DBM-indexed file is used. On both systems, the file content is essentially the same.

The local node name file contains a series of logical records, one for each node name entry in the directory service. Together, these records define each node's fully qualified name, its synonym, and its addresses. This logical record may be looked up using the full name, the synonym, or any of the node's addresses.

Each logical record consists of (1) a node definition physical record, which contains all information related to the node, and (2) zero or more reverse mapping physical records, which contain alternate keys for looking up the node definition. Each reverse mapping record contains only the node name key in its record data. All the data used to describe the node is contained in the node definition record.

Because multiple records compose a node name entry, operations that fail to complete can result in inconsistencies in the local node file. Fortunately, these inconsistencies can be resolved using the same synchronization algorithms as used for DECdns.

CDR API Wrapper Routines for the DECnet Phase IV Node Database

Access to the DECnet Phase IV node database is provided primarily to help users migrate their Phase IV node name data to DECnet/OSI. No access is provided to this database by the CDI library for DECnet/OSI applications. Because this database consists of a simple file, with one record per node name entry, none of the multiple record synchronization problems exist.

Conclusion

The Common Directory Interface, consisting of the CDI registration tool set and the CDI library, provides flexible and extensible directory service access for DECnet/OSI. Initial customer acceptance of these new capabilities has been high and future enhancements are being studied.

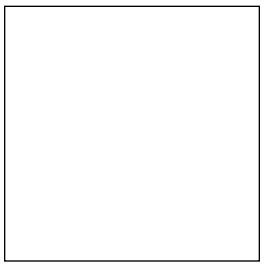
Acknowledgments

The design and development of the Common Directory Interface involved the contributions of the entire directory services and DECnet engineering teams. We extend our thanks to all the team members, as well as to product and engineering management for supporting this project.

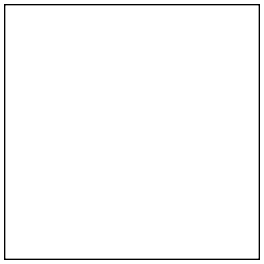
References

1. S. Martin, J. McCann, and D. Oran, "Development of the VAX Distributed Name Service," *Digital Technical Journal,* vol. 1, no. 9 (June 1989): 9–15.
2. P. Mockapetris, "Domain Names—Implementation and Specification," RFC 1035, Internet Document (November 1987).
3. CCITT Sixth Plenary Assembly, "The Directory—Overview of Concepts, Models and Services," *Recommendation X.500 and ISO 9594-1, Data Communications Networks Directory: Recommendations X.500 to X.521, CCITT Blue Book,* vol. xiii.8 (Geneva: International Telecommunications Union, 1989).
4. R. Rosenbaum, "Using the Domain Name System to Store Arbitrary String Attributes," RFC 1464, Internet Document (May 1993).
5. B. Manning and R. Colella, "The Domain Name System NSAP Resource Records," RFC 1706, Internet Document (October 1994).

Biographies



Richard L. Rosenbaum
Rich Rosenbaum is a software engineering consultant in the Internet Software Business Unit, where he is focusing on the application of indexing and collaboration technologies to the World Wide Web. In his 17 years with Digital, he has worked on networking products operating on Digital's 16-, 32-, 36-, and 64-bit platforms. He is the co-author of several patents on network software. Rich obtained a B.S. from the State University of New York at Stony Brook.



Stanley I. Goldfarb
Stan Goldfarb is a principal software engineer with the Internet Software Business Unit. Since joining Digital in 1976, he has contributed to several network and network management projects, including DECnet/RSX, DECnet-PRO, DECnet-DOS, DECmcc, DECnet/OSI, and PATHWORKS, and he has co-authored several patents on network management software. He is currently working on a Workgroup Web Forum application to provide electronic mail subscription and distribution services. Stan holds B.S. and M.S. degrees in computer science from Worcester Polytechnic Institute and an M.S. in management from Lesley College.