CAE Specification

DCE 1.1: Directory Services

The Open Group

CAE Specification

DCE 1.1: Directory Services

Document Number: C705

Published by The Open Group

Any comments relating to the material contained in this document may be submitted to The Open Group at:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

List of Figures

List of Tables

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the *IT DialTone*. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- consolidating, prioritizing, and communicating customer requirements to vendors
- conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- licensing and promoting the Open Brand, represented by the ''X'' mark, that designates vendor products which conform to Open Group Product Standards
- promoting the benefits of the IT DialTone to customers, vendors, and the public.

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

The Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of CAE and Preliminary Specifications through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product. There are currently two forms of Product Standard, namely the Profile Definition and the Component Definition, although these will eventually be merged into one.

The ‘‘X’’ mark is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the X/Open Trade Mark Licence Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on specification development and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

- *CAE Specifications*

  CAE (Common Applications Environment) Specifications are the stable specifications that form the basis for our Product Standards, which are used to develop X/Open branded systems. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a CAE Specification can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. CAE Specifications are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *Preliminary Specifications*

  Preliminary Specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is not a draft specification; rather, it is as stable as can be achieved, through applying The Open Group’s rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a CAE Specification. While the intent is to progress Preliminary Specifications to corresponding CAE Specifications, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

  The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as CAE Specifications, in which case the relevant Technology Specification is superseded by a CAE Specification.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation—programmer's guides, user manuals, and so on—relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the CAE Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.

- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at **http://www.opengroup.org/public/pubs**.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at **http://www.opengroup.org/public/pubs**.

About This Document

This document is a CAE Specification (see above). It specifies the Directory Services in **X/Open DCE**. It is a portability guide for application programs using Directory Services, and a conformance specification for implementations.

Structure

This document is organised into five parts:

Part 1 is an introduction:

- Chapter 1 describes the information model.
- Chapter 2 describes inter-cell operation.

Part 2 defines the application programming interfaces:

- Chapter 3 describes changes to the X/Open Directory Service (XDS) interface.
- Chapter 4 describes changes to the X/Open OSI-Abstract-Data Manipulation (XOM) interface.
- Chapter 5 describes the XDS/XOM convenience functions.
- Chapter 6 describes the Name Service Independent (NSI) interface.

Part 3 defines the Global Directory Service (GDS):

- Chapter 7 lists X.500 services and protocols.
- Chapter 8 is a conformance statement for GDS.

Part 4 defines the Cell Directory Service (CDS):

- Chapter 9 is the service definition.
- Chapter 10 provides the protocol definition.
- Chapter 11 specifes the encodings for the transaction protocol and the solicitation protocol.
- Chapter 12 presents the CDS operations in Interface Definition Language (IDL).

Part 5 contains the appendices:

- Appendix A defines valid characters and naming rules for DCE Directory Service names.
- Appendix B defines object identifiers for CDS attributes.
- Appendix C lists CDS status and error codes.
- Appendix D provides CDS IDL definitions.
- Appendix E contains the GDS structure rule table.
- Appendix F contains the GDS object class table.
- Appendix G contains the GDS attribute table.
- Appendix H contains the header files for the DCE extensions to XDS.

An index is provided.

Intended Audience

This document is written for application programmers who need to make use of directory services, and implementation developers.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — utility names
  - — external variables, such as *errno*
  - — functions; these are shown as follows: *name*( ).
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- The notation [EABCD] is used to identify an error value EABCD.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font.
- Variables within syntax statements are shown in *`italic fixed width font`*.

Trade Marks

Referenced Documents

The following standards and external related documents are referenced in this specification:

ASN.1
: ISO 8824: 1990, Information Technology — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1).

BER
: ISO / IEC 8825: 1990 (ITU-T Recommendation X.209 (1988)), Information Technology — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).

CCITT T.61
: CCITT Recommendation T.61: 1984, Character Repertoire and Coded Character Sets for the International Teletex Service, Geneva, 1980, amended at Malaga-Torremolinos, 1984.

ISO 3166
: ISO 3166: 1988, Codes for the Representation of Names of Countries, Bilingual edition.

ISO 8326
: ISO 8326: 1987, Information Processing Systems — Open Systems Interconnection — Basic Connection-oriented Session Service Definition.

ISO 8327
: ISO 8327: 1987, Information Processing Systems — Open Systems Interconnection — Basic Connection-oriented Session Protocol Specification.

ISO 8327-2 CD
: ISO CD 8327-2: 1990, Information Processing Systems — Open Systems Interconnection — Basic Connection-oriented Session Protocol Specification — Part 2: Implementation Conformance Statement (PICS) Proforma.

ISO 8649
: ISO 8649: 1988, Information Processing Systems — Open Systems Interconnection — Service Definition for the Association Control Service Element.

ISO 8650
: ISO 8650: 1992, Information Processing Systems — Open Systems Interconnection — Protocol Specification for the Association Control Service Element.

ISO / IEC 8650-2 DIS
: ISO / IEC DIS 8650-2: 1990 Information Processing Systems — Open Systems Interconnection — ACSE Protocol Implementation Conformance Statement (PICS) Proforma.

ISO 8822
: ISO 8822: 1988, Information Processing Systems — Open Systems Interconnection — Connection-oriented Presentation Service Definition.

ISO 8823
: ISO 8823: 1988, Information Processing Systems — Open Systems Interconnection — Connection-oriented Presentation Protocol Specification.

ISO / IEC 8823-2 DIS
ISO / IEC DIS 8823-2: 1990 Information Processing Systems — Open Systems Interconnection — Basic Connection-oriented Presentation Protocol Implementation Conformance Statement (PICS) Proforma.

ISO / IEC 9072
ISO / IEC 9072: 1989, Information Processing Systems — Text Communication — Remote Operations — Parts 1 and 2

Part 1: Model, Notation and Service Definition
Part 2: Protocol Specification.

ISO / IEC 9594
ISO / IEC 9594: 1990, Information Technology — Open Systems Interconnection — The Directory, Parts 1 to 8:

Part 1: Overview of Concepts, Models and Services (CCITT X.500)
Part 2: Models (CCITT X.501)
Part 3: Abstract Service Definition (CCITT X.511)
Part 4: Procedures for Distributed Operation (CCITT X.518)
Part 5: Protocol Specifications (CCITT X.519)
Part 6: Selected Attribute Types (CCITT X.520)
Part 7: Selected Object Classes (CCITT X.521)
Part 8: Authentication Framework (CCITT X.509)

ISO / IEC 10021-2
ISO / IEC 10021: 1990, Information Technology — Text Communications — Message-oriented Text Interchange System — Part 2: Overall Architecture (CCITT X.402).

PUB 217
Directory Services PICS Proforma for DAP / DUA (OSTC / DS / DUA / PICS / V1.0) Open Systems Testing Consortium, PUB 217.

PUB 218
Directory Services PICS Proforma for DAP / DSA (OSTC / DS / DSA / PICS / V1.0) Open Systems Testing Consortium, PUB 218.

RFC 1033
M.Lottor, Domain administrators operations guide, 11 / 01 / 1987.

RFC 1034
P. Mockapetris, Domain names — concepts and facilities, 11 / 01 / 1987. (Obsoletes RFC 0973; updated by RFC 1101.)

RFC 1035
P. Mockapetris, Domain names — implementation and specification, 11 / 01 / 1987. (Obsoletes RFC 0973; updated by RFC 1348.)

RFC 1101
P. Mockapetris, DNS encoding of network names and other types, 04 / 01 / 1989. (UpdatesRFC 1034.)

RFC 1348
B. Manning, DNS NSAP RRs, 07 / 01 / 1992. (Updates RFC 1035).

CCITT X.249
CCITT, 1992, Data Communication Networks: Open Systems Interconnection (OSI), Series X Recommendations (X.220 to X.290), Draft Recommendation X.249 — Remote Operations Service Element: Protocol Implementation Conformance Statement (PICS) Proforma.

The following X/Open documents are referenced in this specification:

DCE DFS
: Preliminary Specification, September 1996, DCE 1.1: Distributed File Service Specification, (P409).

DCE RPC
: CAE Specification, August 1994, X/Open DCE: Remote Procedure Call (ISBN: 1-85912-041-5, C309).

  This specification is now also ISO International Standard ISO/IEC 11578:1996, Information technology — Open Systems Interconnection — Remote Procedure Call (RPC)

DCE Security
: Preliminary Specification, April 1996, X/Open DCE: Authentication and Security Services (ISBN: 1-85912-013-X, P315).

DCE Time
: CAE Specification, January 1994, X/Open DCE: Time Services (ISBN: 1-85912-067-9, C310).

X.400, Issue 3
: Also known as XMHS Issue 3: CAE Specification, May 1996, API to Electronic Mail (X.400), Issue 3 (ISBN: 1-85912-185-3, C609).

XDS, Issue 3
: CAE Specification, May 1996, API to Directory Services (XDS), Issue 3 (ISBN: 1-85912-180-2, C608).

XOM, Issue 3
: CAE Specification, May 1996, OSI-Abstract-Data Manipulation API (XOM), Issue 3 (ISBN: 1-85912-175-6, C607).

CAE Specification

Part 1

Introduction

The Open Group

Chapter 1

Information Model

The *directory* is a collection of information about a set of objects. Objects are referenced in *entries* of the directory service. Entries that are structured recursively in the system are called *directories*. Information in directory service entries can be used for purposes such as describing resources, location of the referenced resources, and for discovery of operations on objects.

While it is conceivable that a directory service may be built on a general purpose database, the directory service is not conceptually such a database. The distinct characteristics of the directory service information model are best described by the following assumptions:

- The underlying organisational structure is hierarchical.
- Queries are the dominant operations, resulting in a high ratio of look-up operations in comparison to update operations.
- Due to the relatively slow rate of changes, instantaneous reflection of updates is usually not required (that is, the state of transient conditions may be acceptable).

This document, which is the specification of the directory service information model, assumes the existence of the above generic characteristics. However, the specification may be applied to a wide variety of services, ranging from general purpose directory services to very specialised resource managers. Information storage in logical databases is common to all of these, but this specification does not specify the properties (such as insertion and retrieval speed, partitioning, replication) of these information bases. The definition of these properties is subject to service-specific specifications.

Directory entries contain *attributes*. Attributes are discrete items of separately accessible information associated with a directory object. Each attribute is determined by its type, and its value (or set of values) represents an instance of this class of information. Depending on the specific semantics of directory services, the representation and richness of attributes varies.

The directory service specifications define the possible operations on directory entries. However, the most rudimentary common operations, supported by every directory service, are **read**, **modify**, **create** and **delete** (on entries and attributes).

1.1 Names and Identifiers

A *name* is a string of characters that refers to an object. Directory entities are identified by names. A name resolves a directory entry *unambiguously*, but does not provide for uniqueness. Multiple names can reference the same entry, and names can be *aliases* or soft links. An alias expresses an alternative name of an object, representing a possibly different hierarchical relationship.

An *atomic name* represents the *distinguished value* of a particular entry. The ingredients of an atomic name are determined by the directory service's rules and semantics. An atomic name can be:

Untyped
An unambiguous string of characters determined only by its entry's position in the organisational structure; for example, its hierarchy level

Typed
A set of designated attribute type-value pairs, represented by multiple attribute-value-assertions, or *AVA*s

A *relative distinguished name* is a typed atomic name describing the distinguished value of an object entry. The relative distinguished name of an object entry is unique (in its context, relative to its parent's name) and must not express an alias.

A sequence of one or more atomic names that represent entries in a particular name service is a *compound name*. The rules for the composition of compound names are determined by the name service's name syntax. A POSIX pathname, for instance, is a compound name that describes an entry in the file system.

The rules and conventions for the DCE-defined compound names are specified in Section 1.3 on page 11 of this document. The compound names specified there are *global compound name* and *cell compound name*.

A compound name consists of distinguished *components* that are equivalent to atomic names in the global name space (for X.500) and the cell name spaces. Internet DNS names in the global name space consist of exactly one component that is equivalent to the compound name, where a component is a set of . (dot) separated atomic DNS names. The contents and parsing rules for a component are defined by the semantics of the name space of which the component is a member.

Components may represent a single physical directory or a leaf object; they may be typed or untyped; or they may represent complex structures such as attribute-based schemes. The present document only specifies syntax and semantic rules for components in the global and cell name spaces, as outlined in Section 1.3 on page 11. The semantics and syntax rules for other composite name spaces is outlined in the corresponding documents (see Section 1.2.3 on page 10 for specific references).

A *composite name* consists of one (non-empty) or multiple consecutive compound names. A composite name may thus represent multiple name services, with possibly distinct syntactic and semantic rules. Composite names adhere to the conventions outlined in Section 1.3 on page 11.

Entities in the name system are identified by their *name context*. A name context has an associated name convention, expressed by the semantic and syntax rules of the controlling name service. The scope of a composite name is defined by its initial name context. This document defines the following three initial contexts:

Global context
The single global root of the DCE directory information system.

Cell context
The context of the root of the local cell (refer to **DCE Cell** on page 9 for the definition of cell).

File context
The root of the distributed file system within the local cell.

An atomic name is unambiguous only within a given name context (relative to its immediate superior, the parent directory entry). However, a *fully-qualified global name* characterises an entry unambiguously within the global context. A fully-qualified global name consists of an ordered set of compound names, beginning from the global root.

The *distinguished name* of a directory service entry is a fully-qualified global name, *uniquely* identifying the object (no aliases).

The uniqueness of a given directory entry may also be expressed by an *identifier*. Identifiers are unique in a given context. Contexts are determined by the directory service and the type of identifier used. This document specifies two types of identifiers: the OSI Object Identifier (OID), and the Universal Unique Identifier (UUID).

*OID*s (object identifier) are obtained from a hierarchy of allocation authorities, the highest being the International Organisation for Standardization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT). The printable format of object identifiers is a string of . (dot) separated digits (see also the ASN.1 standard).

The *UUID* (universal unique identifier) is a fixed size identifier which is guaranteed to be unique across both space and time. A further specification of the UUID can be found in the **DCE Remote Procedure Call** specification.

1.2 Name Spaces

A *name space* is an organisational model that describes the syntactic and semantic rules within a particular domain. The concrete operations allowed within a name space are provided by one or multiple associated name services that control the name space. Name spaces are hierarchically ordered, and name space boundaries are crossed through *next naming system pointers* (see **Next Naming System Pointers** on page 8).

Note: The notion of name spaces is derived from the conceptual model commonly known as *federated naming*. Federated naming prescribes a system of heterogeneous naming technologies that are interworking through bilateral cooperation.

The DCE directory information model consists of a set of composite name spaces. Composite name spaces are linked together in a hierarchical order. This document specifies two types of composite name spaces: the global name space and the cell name space (see Figure 1-1).

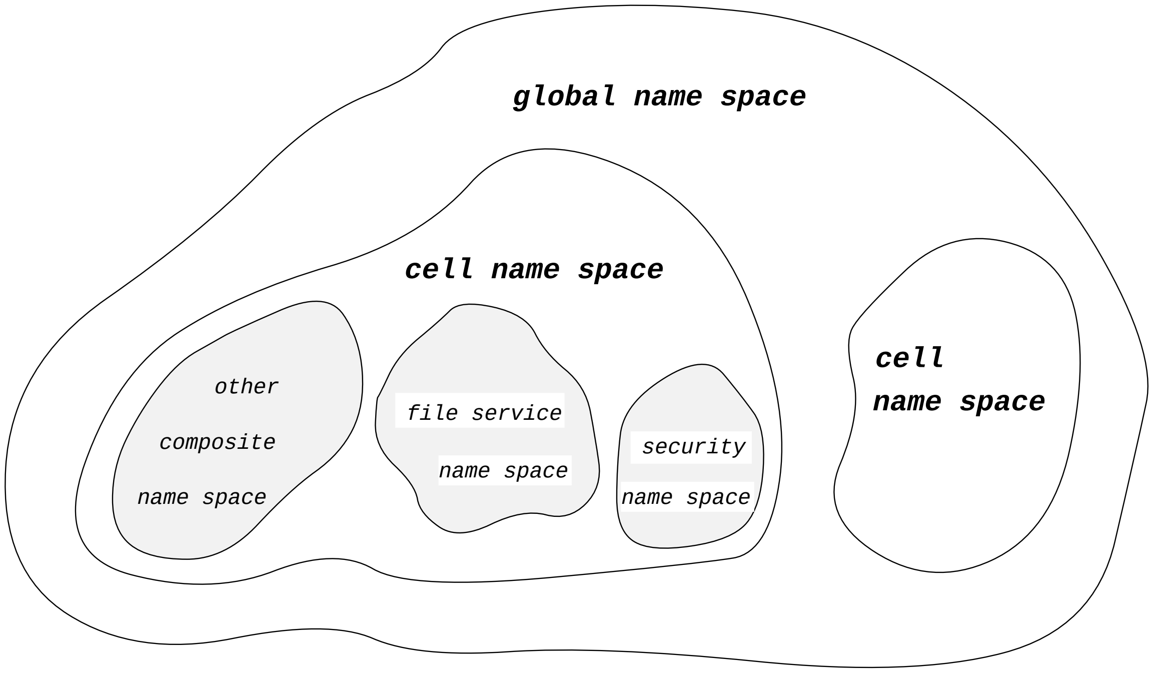

Figure 1-1 DCE Name Spaces

The *global name space* is at the top of the hierarchical arrangement of name spaces. The global name space, and therefore implicitly the entire DCE directory information system, is determined by the global context, expressed by the global root.

Particular entries within the global name space identify *cell name spaces*; these cell name spaces are subordinates of the global name space. Cell name spaces can also be subordinate to other cell name spaces in a configuration called a *cell hierarchy*. The top-level cell in a cell hierarchy, the *parent cell*, is always catalogued in the global name space. The name spaces of the *child cells* are subordinates of the name space of the parent cell.

Cell name spaces, then, are either subordinates of the global name space or of other cell name spaces. The root and attributes of the top-level cell name space are catalogued in the global name space. The root and attributes of child cells are catalogued in the parent cell name space. Either parent cell name spaces or child cell name spaces can be superordinates for other composite name spaces (see Section 1.2.3 on page 10).

The entities in both the global and the cell name spaces are hierarchically structured, and together comprise an inverse tree of nodes. These name spaces represent an acyclic directed graph: each entity has exactly one parent at any moment, and no entity is a descendent of its own descendents. This architecture does not impose the structural rules applied to other composite name spaces.

1.2.1 Global Name Space

The global name space, which is the top-level name space of the DCE directory information model, provides for a universally unique root and contains cell name spaces as subordinates. The global name space, as specified here, is represented by two directory services:

- the international standard Directory Service CCITT X.500 or the ISO 9594 standard
- the Internet Domain Name Service (DNS) (see Internet RFC 1101, RFC 1035, RFC 1034 and related documents).

The syntax and semantics of the global name space are in accordance with the standard specifications of these directory services. The conformance requirements are specified in this document.

1.2.2 Cell Name Space

The *cell name space* consists of hierarchically organised directory service entries of untyped atomic names. Entries are unambiguously identified by their name and position within the hierarchy, and are (optionally) uniquely identified by a UUID. The cell name space rules do not enforce any particular model for organising the entries in the name space.

The cell name space defines three different types of entries[1]:

Directory

Directory entries represent nodes in the name space tree which contain arbitrarily many references to subordinate entries (objects, soft links, and pointers to other directories).

References to descendent directories of a particular directory entry are called *child pointers*. Child pointers link directories together into a rooted tree, in which there is a single path from the root directory through a set of child directories, to the desired named object. Child pointers are created internally by the directory service.

Directories can be empty, not containing any references to subordinate entries.

Every directory entry has exactly one ancestor directory. With one exception, every directory uses a parent pointer to reference its ancestor. The top-level directory entry of the top-level cell is the exception; in that case, the directory entry uses a global pointer to reference the global name space.

Directory entries do not have parent pointers to soft link entries (the directory name to which a parent pointer refers is a fully-qualified global name).

To prevent cycles in the tree structure of the cell name space, a directory must not be a child of any of its descendents.

1. Note that the meanings of the terms *directory* and *object*, as used in this document, vary with the context in which they are used. In the context of a cell name space, the terms refer to particular entries within that name space; in the context of the global name space, a directory represents a set of entities.

Object

Object entries represent terminal or leaf nodes in the cell name space.

Note: These leaf entries usually do not represent the target object of an operation itself but the catalogued name of an object. These entries may control object references and other information on behalf of the target object. DCE uses these object entries for registering the binding information (object references) of servers and subordinate name spaces.

Soft link

A soft link entry is a form of alias or indirect pointer that provides an alternate name for either an object, directory, or another soft link entry. Soft links thus allow a single entry to be reached via more than one name.

The contents of every cell name space entry consist of a set of attributes and their associated values.

Attributes are of two types:

single-valued

A single-valued attribute can have only one value at a time.

set-valued

A set-valued attribute can have more than one value at a time. Its contents consist of a set of unordered values, called *members*. These values must be distinguishable; hence, set-valued attributes cannot have duplicated members.

From the point of view of usage, attributes can generally be divided into two categories:

operational

Operational attributes are those which are predefined by the Cell Directory Service (CDS).

application-specific

Application-specific attributes are those attributes which can be freely attached to object and directory entries by applications or users, assuming that they have the appropriate access privileges. (Note that soft link entries are excluded from this category.)

The set of these categories of attributes, and their applicability to distinct entries (directory, object, soft link), is service and implementation-specific.

An attribute is identified by an *attribute identifier* (sometimes referred to as *attribute name*), which distinguishes it from other attributes. An attribute of a given name can have only one attribute value syntax (data type); it is not possible for a single-valued attribute and a set-valued attribute to have the same name at the same time. Every attribute name has an OID (Object Identifier) associated with it to ensure uniqueness among attributes.

Next Naming System Pointers

Name spaces are linked by *next naming system pointers*. Next naming system pointers accomplish the concept of obtaining the "next name space" reference.

The protocols used to resolve the next name space reference pointers are not specified in this document (though it is anticipated that such protocols will be supported in the future).

The canonical string representation of DCE names does not make a syntactical distinction between atomic name and compound name separators for the cell name space (both use / (slash) as the separator; see Section 1.3 on page 11). The boundaries between cell and subordinate name spaces can only be determined by resolution. Therefore, an (untyped) name of a cell name space entry may collide with a top-level name of a subordinate name space.

To avoid these ambiguities in a composite name, it is recommended that next naming system pointers always be object (leaf) entries in the cell name space, which explicitly associates a name to the name space pointer. These *explicit* next naming system pointers are called *junctions*. This specification only supports junctions in the cell name space.

Note: This restriction does not apply to cell registration in the global name space, which can be catalogued in any node within the global name space (global name spaces controlled by DNS have distinct atomic name separators and those controlled by GDS contain typed names). This permits the use of the same name for both a pointer to another name space and to maintain other descendants within the same name space. These *implicit* next naming system pointers can only point to a single subordinate name space.

As defined in Section 1.3.2 on page 13, a composite name may consist of multiple compound names — the global compound name, the cell compound name, and a residual that identifies entities in subordinate composite name spaces. Each of these is a set of one or more name components, separated by a / (slash).

If the rightmost component of a name that resides in the cell name space is *not* the rightmost component of that composite name, the resolved entry represents a junction. The content of the residual compound names of the composite name is opaque to the cell name space directory service.

Cell name space junctions to subordinate name spaces are catalogued (that is, have a name) in the cell name space. Pointers to and information about the resource manager of a subordinate composite name space are registered in application-specific attributes of the junction entry.

The junction entry represents both the entry in the cell name space and the object of the subordinate name space that this junction is bound to. This bound object is usually the root of that subordinate name space.

This document does not specify how the cell name space junction entry and the bound object can both be unambiguously accessed. Applications may provide programming interfaces that disambiguate the access to these entities that are referred to by the same name. The ACL Editor, for instance, defines a flag in the *sec_acl_bind*( ) interface that allows applications to determine the target of the operation (the ACL of the cell name space entry or the ACL of the protected object of a server-supported name space) — see the **DCE Security Services** specification.

DCE Cell

A DCE *cell* is an administrative and security domain within the cell name space. Servers of the Cell Directory Service (CDS) and security authorities are instantiated on a cell-wide basis.

A cell is typically a group of users, systems and resources that are centred around a common purpose and which share common DCE services. A minimum cell configuration includes one CDS, one Security Service and one Time Service.

Because cells are administrative units, organisational notations such as the shorthand for the cell root, **/.:** (see Section 1.3.2 on page 13), are related to the cell and not the cell name space. This may not be visible if a cell name space consists of exactly one cell, but if hierarchies of cells are introduced within a cell name space, the cell root always refers to the root of the local cell.

The membership of a cell is determined on a per host basis. How a host obtains information about its cell identity is implementation-specific; for example, the cell name may be stored in a local (host-specific) configuration file. The local cell is the cell defining the environment of the initial user's login session.

1.2.3 Other Composite Name Spaces

This document does not specify the particular semantics and syntax rules for composite name spaces other than the global and the cell name space. However, the **DCE Security Services** specification specifies the *security name space*, and the **DCE File Services** specification specifies the *file service name space* (also called *filespace*). Both name spaces are subordinates of the cell name space, and are integrated through junctions.

Within each cell, the actual names for these junctions are not determined by the architecture; they are indirectly identified in the **/.:/cell-profile** profile entry as having the name associated with the UUID of the junction. However, the recommended names are **/.:/sec** for the security name space, and **/.:/fs** for the file service name space.

Other name service or resource manager implementations may be integrated into the DCE name space, in accordance with the rules outlined in this specification.

1.3 Name Syntax

Names defined by this syntax consist of strings of characters (elements of a character set). The minimal set of valid characters is defined in the *Portable Character Set* (PCS) for the DCE directory information system. Figure A-1 on page 172 represents this portable character set and its applicability to the name spaces.

The *PCS* for directory services defines the semantics of names, rather than the actual visual or encoded characters. For example, it specifies that a character with the semantic of backslash (that is, the \ character) must be supported, although in some fonts the backslash glyph may have been replaced with another glyph. Note that a similarity in visual appearance between two glyphs does not necessarily mean that they are in fact the same semantic character.

Characters are internally associated with an opaque *encoding*; that is, one or more numerical values and bit patterns. Directory service specifications for GDS and CDS specify the internal representation of names (see Parts 3 and 4, and associated appendices).

Neither varying character sets (for example, for native language support), other than the portable character set, nor different visual presentations (such as input through a graphical user interface) are specified in this document.

The syntax applied when interpreting a name is dependent on whether a component refers to an entry in the global name space, the cell name space, or another composite name space. The common characteristics of the syntax are specified as follows:

The characters / (slash) and \ (backslash) have special properties, and are called the *metacharacters* of the syntax. Additional distinct metacharacters for both the global and the cell name space are defined in respective sections of this document. Their special properties can be cancelled by preceding them with a \ (backslash), an operation called *escaping*. The sequence of characters consisting of a \ (backslash) followed by an ordinary character (that is, by a non-metacharacter) is reserved, and may be specified in subsequent versions of the following syntax.

1.3.1 DCE Name Syntax

Backus-Naur Format (BNF) of DCE Name Syntax on page 12 defines the DCE name syntax. The sections that follow the table explain how the syntax is applied to the different DCE name spaces and services. References in the text to elements defined in the BNF are in the form: <*element*>.

The notations used are as follows:

`::=`	Is defined to be.
`|`	Alternatively.
`<text>`	Non-terminal element.
`"`	Literal expression.
`*`	The preceding syntactic unit can appear 0 or more times.
`+`	The preceding syntactic unit can appear 1 or more times.
`{}`	The enclosed syntactic units are grouped as a single syntactic unit (can be nested).

Backus-Naur Format (BNF) of DCE Name Syntax

The DCE name syntax is as follows:

```
PCS                 ::= { Portable Character Set }
SimpleChar          ::= { Subset of symbols from Portable Character Set,
                          intersection of characters supported by
                          GDS, DNS and CDS -
                          a to z, A to Z, 0 to 9, "-" }
AlphaChar           ::= { a to z, A to Z }
NumChar             ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

WildChar            ::= ? | *
EscapeChar          ::= \
ComponentSep        ::= /
DotSep              ::= .
Quote               ::= " | ’

GlobalContext       ::= <ComponentSep>+ "..."
CellContext         ::= <ComponentSep>+ ".:"
FileContext         ::= <ComponentSep>+ ":"

DNSChar             ::= <SimpleChar>
DNSAtomicName       ::= <DNSChar>+
DNSComponent        ::= <DNSAtomicName> {<DotSep> <DNSAtomicName>}+
DNSCompoundName     ::= <DNSComponent>

GDSAVASep           ::= =
GDSMultiAVASep      ::= ,
GDSChar             ::= <SimpleChar> | <DotSep>
                        | " " | ‘ | ( | ) | + | : | ?
GDSAttrType         ::= {<AlphaChar> {<AlphaChar> | <NumChar>}*}
                        | {<NumChar>+ {<DotSep> <NumChar>+}*}
GDSAttrValue        ::= <GDSChar>+
                        | {<GDSChar>* <EscapeChar> <PCS> <GDSChar>*}+
                        | <Quote> <PCS>+ <Quote>
GDSAVA              ::= <GDSAttrType> <GDSAVASep> <GDSAttrValue>
GDSAtomicName       ::= <GDSAVA> {<GDSMultiAVASep> <GDSAVA>}*
GDSComponent        ::= <GDSAtomicName>
GDSCompoundName     ::= <GDSComponent> {<ComponentSep>+
                        <GDSComponent>}*

GlobalCompoundName  ::= <DNSCompoundName> | <GDSCompoundName>
GlobalCompositeName ::= <GlobalContext> <ComponentSep>+
                        <GlobalCompoundName> | <GlobalContext>
                        <ComponentSep>*

CDSChar             ::= <GDSChar> | <GDSMultiAVASep> | ! | # | $ | %
                        | & | ; | < | > | [ | ] | ˆ | _ | { | }
                        | "|" | ˜ | *
```

```
CDSAtomicName       ::= {<CDSChar> | <GDSAVASep>}+
                        | {{<CDSChar> | <GDSAVASep>}* <EscapeChar> <PCS>
                          {<CDSChar> | <GDSAVASep>}*}+
                        | <Quote> <PCS>+ <Quote>
CDSFirstComponent   ::= <CDSChar>+
                        | {<CDSChar>* <EscapeChar> <PCS> <CDSChar>*}+
                        | <Quote> <PCS>+ <Quote>
CDSComponent        ::= <CDSAtomicName> | <CDSFirstComponent>
CDSCompoundName     ::= <CDSFirstComponent>
                        {<ComponentSep>+ <CDSAtomicName>}*

CellCompositeName   ::= <CellContext> <ComponentSep>+ <CDSCompoundName>

AtomicName          ::= <SimpleChar>+
                        | {<SimpleChar>* <EscapeChar> <PCS>
                           <SimpleChar>*}+
                        | <Quote> <PCS>+ <Quote>
CompoundName        ::= <AtomicName> {<ComponentSep>+ <AtomicName>}*

FullName            ::= <GlobalCompositeName>
                        {<ComponentSep>+ <CDSCompoundName>}*
                        | <GlobalCompositeName>
                          <ComponentSep>+ <CDSCompoundName>
                          {<ComponentSep>+ <CompoundName>}*
CellRelativeName    ::= <CDSCompoundName>
                        {<ComponentSep>+ <CompoundName>}*
FileName            ::= <FileContext> <ComponentSep>+ <CompoundName>
```

1.3.2 DCE Composite Names

This section defines the policies that are applied to composite names in DCE.

A DCE composite name is structured as a left-to-right ordered set of compound names, and a compound name is structured as a left-to-right ordered set of components. This ordering is intended to correspond to the top-to-bottom (parent/child, superordinate/subordinate, more-significant/less-significant, big-endian/little-endian) lineage of the named object.

The ordering of compound names is:

global compound name, followed by
a cell compound name, followed by
compound names of other composite name spaces (including names of child cells within a hierarchical cell configuration).

These compound names represent entries in their associated name spaces.

Multiple compound names and components of compound names are separated by / (slash) characters (BNF element <ComponentSep>).

Note: The name syntax permits empty component and compound names — multiple consecutive occurrences of the component separator / (slash). A trailing component separator implies a trailing empty component. These empty components, however, will be eliminated during canonicalisation and not further processed (see **Canonicalisation of Names** on page 14).

Initial contexts are special cases of reserved component names. They are logical entities describing the name context; they do not represent actual entries in the directory services. The initial contexts are separated from the remainder of a composite name by a / (slash). The three defined initial contexts are:

- Global context (BNF element <GlobalContext>)
- Cell context (BNF element <CellContext>)
- File context (BNF element <FileContext>).

A fully-qualified global name (BNF element <FullName>) is a DCE composite name with the global root at the top (left). Hence, the fully-qualified global name is a single path in a hierarchically structured rooted tree of name spaces. The name of the global root is represented by the character string **/...**. Fully-qualified global names must begin with the substring **/...**.

The name of the cell root is **/.:**. This is the shorthand form for the fully-qualified global name of the local (or current) cell. If the cell is registered in global name space, **/.:** resolves into */.../{global compound name}*. If the cell is registered in a cell name space (that is, the cell is a child cell within a cell hierarchy), then **/.:** resolves into */.../{global compound name}/{child cell name}*, where *{global compound name}* refers to the parent cell. The **/.:** cell context provides a syntactical convenience for establishing a composite name, but does not change the name semantics, except that a given cell compound name (BNF element <CDSCompoundName>):

- when applied to a different cell, may resolve into an object different from that registered in the cell in which the name was meant to be applied
- when applied to a child cell within a cell hierarchy resolves to the cell root of the child cell, not the cell root of the parent cell.

In names specified from the global root, the name for the cell root is always placed to the left of the rest of the cell name, as a prefix to the composite name (BNF elements <CellCompositeName> and <CellRelativeName>).

Similarly to the cell root, the file context is a shorthand form for the cell relative name of the local distributed file service. The name of the file context is **/:**, which resolves to **/.:/**<string> (where the recommended name for <string> is **fs**).

Canonicalisation of Names

The name syntax defines the rules for names that are to be presented to the name service interfaces. Before a name is processed by a name service, and before any of the matching rules specified in Section 1.3.3 on page 15 are applied, the name is converted to its most primitive, *canonical* form. This process is called *canonicalisation*; its successful completion implies that the name is syntactically correct, according to the syntax rules that are outlined below.

Canonicalisation is performed according to the following rules:

1. Multiple successive / (slash) characters are reduced to one slash; that is, empty components are eliminated and will not be processed.
2. Similarly, trailing / (slash) characters are eliminated and will not be processed.
3. If a typed name component, representing an X.500 RDN (Relative Distinguished Name, BNF element <GDSComponent>), is encountered:

   — Multiple successive space characters are reduced to a single space, regardless of the applicable case matching rules, unless the name is quoted.

- — Leading space characters and spaces between an attribute type and an = (equal sign) character are removed.
- — If the matching rules for the attribute type specify the value to be case insensitive, all alphabetic characters in the value string are converted to lower case.
- — If the attribute type is determined to be in object identifier syntax:
  - — it is converted into an object identifier representation (that is, a . (dot) separated string of digits; note that the mapping from tag string to object identifier is based on Table G-1 on page 209).
  - — otherwise, and if the case matching rule for the attribute type is determined to be case insensitive, the attribute type is converted to an all lower-case tag string.

4. If the name component represents an Internet DNS compound name (BNF element <DNSCompoundName>), the component name string is converted entirely to lower case.

1.3.3 Global Name Space Name Syntax

The , (comma) and = (equal sign) characters are *metacharacters* in the global name space syntax. As with the other valid metacharacter (that is, the / (slash) character), these may be escaped by a preceding \ (backslash) character. Escaping the backslash character itself (that is, having multiple contiguous occurrences of \) within the global compound name is reserved for future use, and is not allowed by the current version of the specification.

The global name syntax specified in this document distinguishes between the two supported global directory services, X.500 and DNS. An X.500 component in a fully-qualified global name is precisely and immediately identified by the occurrence of at least one = (equal sign) character in the first name component after the global root (/...); the equal sign indicates the presence of an attribute-value-assertion (AVA) in a GDS compound name (BNF element <GDSCompoundName>). (See below, **X.500 Typed Names**.)

If the first component of the global compound name (the descendent of the global root **/...**) is not identified as an X.500 name (that is, it does not contain a = (equal sign)), and if it contains at least one . (dot) character, it is considered to be a DNS name. Names that are not identified as being either X.500 or DNS are not specified (BNF element <GlobalCompositeName>). Implementations may reject these as unsupported.

X.500 Typed Names

An X.500 global compound name (BNF element <GDSCompoundName>) may consist of arbitrarily many components. Every name component represents a relative distinguished name (RDN) according to the X.500 rules.

A component (BNF element <GDSComponent>) consists of a set of arbitrarily many (but at least one) *subcomponents*, each of which represents an attribute-value-assertion (AVA). Subcomponents are separated by the , (comma) metacharacter. The ordering of subcomponents is significant for applying the matching rules, but the rules for ordering subcomponents are application determined. Subcomponents within a component need not be distinct. Subcomponents cannot be empty. Multiple successive , (comma) characters are not allowed. Similarly, components cannot end with a comma.

Every subcomponent (BNF element <GDSComponent>) must contain exactly one = (equal sign) character. The substring preceding the = is called the *type*, and the substring following is called the *value*. Types and values are non-empty character strings, containing no unescaped metacharacters, and may be arbitrarily long. Since types and values cannot be empty, multiple successive = (equal sign) characters are not allowed. Similarly, components cannot end with an equal sign. In other words, within a component, the number of unescaped equal signs matches exactly the number of comma characters plus 1.

Attribute types (BNF element <GDSAttrType>) must begin with an alphabetic character, can contain alphanumerics, and cannot contain spaces.[2]

Two global compound names which *match* each other according to the following rules are interpreted to always name the same object[3]:

- Two global compound names match if and only if all their components match, in sequence (left-to-right).
- Two components match if and only if they match as ordered sets of subcomponents. The rules for ordering subcomponents are not relevant: two logically equivalent components that refer to the same name service entry do not match if their subcomponents are provided in a different order. Subcomponents may be identical, but they match if and only if their order of occurrence matches.
- Subcomponents match if and only if their attribute types and their attribute values both match.
- Attribute types are matched case insensitively, character by character.
- Attribute values can be matched case exactly or case insensitively, depending on the rule defined for its type at the DSA (Directory Service Agent).

DNS Names

The global compound name in DNS consists of exactly one component, which follows the global root prefix. The full DNS name, formed in accordance with its syntax rules (that is, consisting of domains separated by . (dot) characters, in little-endian ordering, case insensitive, and so on), is represented in this single component (BNF elements <DNSCompoundName> and <GlobalCompoundName>, separated by a / (slash) character).

A global DNS name (BNF element <DNSComponent>) referencing a cell name space must contain at least one . (dot) character (that is, must be more than one level deep), and may consist of multiple domain names, each separated by a . (dot) character.

Two global compound names which *match* one another according to the following rule are interpreted to always name the same object:

- Two global compound names are matched case insensitively, character by character.

2. An alternative method of specifying attribute types is by OSI Object Identifiers (a string of digit groups, separated by . (dot) characters).

3. It is important to recognise that the matching rules specified here define an *equivalence relation* on the name space, which is different from the notion of *string equality* of names (character-by-character equality). Also, note that these are matching rules *for the name syntax itself,* not for the named object. In particular, two non-matching names may name the same object in a given implementation.

1.3.4 Cell Name Space Name Syntax

The * (asterisk) and ? (question mark) characters are *metacharacters* (BNF element <WildChar>) for the cell name space syntax. Any valid metacharacter may be escaped by preceding it with a \ (backslash) character.

A component consists of a character string representing the (untyped) atomic name of an entry. The ordering of components (left-to-right) represents the hierarchy (top-to-bottom) of the cell name space organisation (BNF elements <CDSComponent> and <CDSCompoundName>).

The * (asterisk) metacharacter acts as a wildcard character matching zero or more characters. The ? (question mark) metacharacter acts as a wildcard character matching exactly one character. Both * and ? are interpreted as metacharacters only when they occur in the rightmost component of a composite name. The CDS specification determines how operations interpret these wildcard metacharacters. For example, a look-up operation may return only the first matching entry or may return all matching entries; modify operations may reject the request; and so on (see Section 9.1.1 on page 87).

The first component (BNF element <CDSFirstComponent>) in the cell compound name must not contain the = (equal sign) character, unless it is escaped by a preceding \ (backslash) metacharacter.

Two cell compound names which *match* one another according to the following rules are interpreted to always name the same object:

- The global compound names match (if the cell context **/.:**) is used, it must resolve to the same cell name space).
- Two cell compound names are matched case exactly, character by character (wildcard matching is implementation-specific).

Hierarchical Cell Names

Within a cell hierarchy, the fully-qualified global name of a child cell consists of the fully-qualified global name of the parent cell with the CDS name of the child cell appended to it.

In all cases, the cell context (**/.:**) resolves to the fully-qualified global name of the current cell, whether that cell is a parent cell (registered in the global name space) or a child cell (registered in the cell name space of the parent cell). Thus, when resolved from within a child cell, the cell context (**/.:**) refers to the root of the child *not* to the root of the parent.

Cell Aliases

A cell can be known by (and answer to) different names, but only one can be designated as the *primary cell name*. This is what the cell context (**/.:**) will translate to within a given cell, and is the name that DCE services return. Alternate cell names, called *cell aliases*, allow cells to be registered in more than one global namespace, and provide the means by which a cell's name can be changed.

1.4 Conformance Requirements

To conform to this document, implementations must meet the following requirements:

- Implementations must conform to the rules for naming and syntax specified in this chapter and in Section A.1 on page 173, Section A.5 on page 180, and Section A.6 on page 181.
- Implementations must conform to the semantics and data formats specified in Chapter 2.
- XDS implementations must follow the conformance rules specified in Section 3.1 on page 28 and Section 4.1 on page 45.
- XDS implementations must support the character encodings as specified in Section A.3 on page 177 and Section A.4 on page 178.
- GDS implementations must follow the Protocol Implementation Conformance Statement specified in Chapter 8.
- GDS implementations must support the structure rules and the associated object class and attribute tables specified in Appendix E, Appendix F, and Appendix G.
- CDS implementations must adhere to the rules and the semantics specified in Section 9.1 on page 87 and Section 9.2 on page 88.
- CDS implementations must support the architected default attributes specified in Section 9.3 on page 96 and Section B.2 on page 186.
- CDS implementations must conform to the service primitives specified in Section 9.4 on page 106.
- CDS implementations must support the protocol operations specified in Chapter 10 and Chapter 12.
- CDS implementations must conform to the ordering of protocol operations as specified in Section D.1 on page 192 and Section D.2 on page 197.
- CDS implementations must support the protocol encodings specified in Chapter 11.
- CDS implementations must support the specified size values in Section A.7 on page 183.
- CDS implementations must support the status and error code encodings specified in Appendix C.

Chapter 2

Inter-cell Operation

This chapter describes inter-cell name space communication. There are two modes of inter-cell communications:

Inter-cell Communication via the Global Name Space

Standard inter-cell communication is accomplished via the global name space, where the Cell Directory Service (CDS) that controls a cell's name space is registered. These cell entries in the global name space consist of information necessary for locating and accessing the CDS of a cell. The Global Directory Agent (GDA) acts as the gateway for obtaining the location information for a foreign cell's CDS from the appropriate global directory service (see Figure 2-1 on page 20).

Hierarchical Cell Communications

In a *hierarchical cell configuration*, one or more cells can be registered in another cell's name space. The *parent cell*, at the top of the hierarchy, must be registered in a global name space, but the *child cells* are registered in the parent cell's name space. In a hierarchical cell configuration, the Global Directory Agent (GDA) is again the gateway for obtaining a child cell's location information; in this case, the GDA obtains the location information from the CDS of the parent cell instead of from a global directory service.

2.1 Global Directory System Organization

A Cell Directory Service (CDS) uses the location information in the global name space to connect to the name spaces of foreign cells (that is, cells that are not in a hierarchical relationship with the requesting cell). The relationship of independent, non-hierarchical cells registered in global name spaces is illustrated in Figure 2-1.

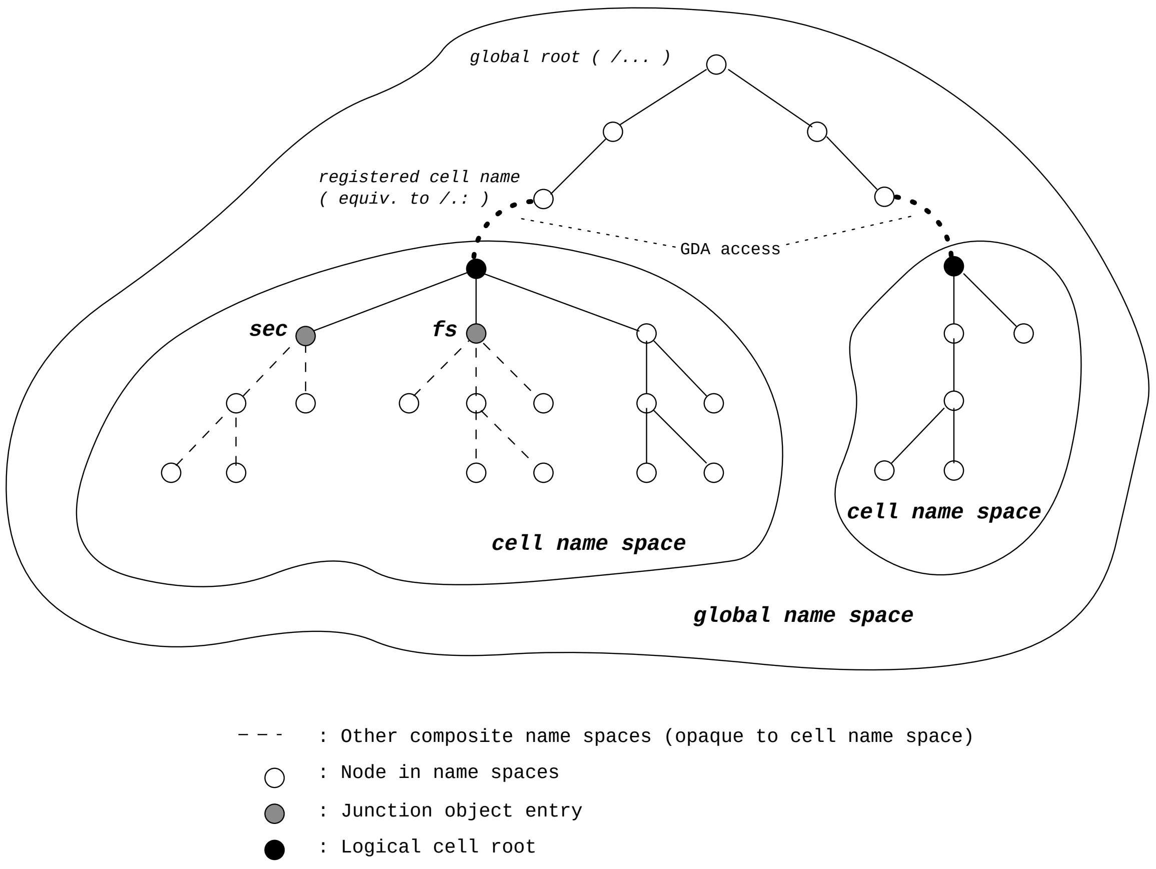
global root (/...)
registered cell name (equiv. to /.:)
GDA access
sec
fs
cell name space
cell name space
global name space
- - - : Other composite name spaces (opaque to cell name space)
○ : Node in name spaces
● : Junction object entry
● : Logical cell root

Figure 2-1 Directory System Organization

2.2 Global Directory Agent

The *Global Directory Agent* (GDA) is the gateway between the cell and global name spaces; it queries the desired foreign cell name space information (that is, the location and protocol data of the targeted CDS) registered in the global name space. The GDA process resolves the global compound name of any fully-qualified global name (prefixed by the global root **/...**) that contains a foreign cell name space unknown to the local cell name space. This information is contained in the lookup *progress record* returned by the GDA.

The GDA parses the global compound name according to the syntax rules for DCE names. If the global compound name is identified as an X.500 name (that is, if it contains at least one = (equal sign) character in its top-level component), the GDA extracts the global compound name from the fully-qualified global name, discards the global root component, and issues a request to the corresponding X.500 service.

If the global compound name is not identified as being an X.500 name, the GDA assumes that the global compound name is an Internet DNS name, and passes the request to the DNS service. Since the DNS global compound name always consists of exactly one component (with . (dot) character separated subcomponents), only the first component of the unresolved name will be extracted for the request issued to the corresponding DNS server.

The *progress record*, which is returned by a successful[4] GDA request, contains information about the directory service of the targeted cell name space. It also contains the unresolved *residual* of the composite name (see Section 2.3 on page 23). This information can be used by the local CDS to locate and access the targeted foreign CDS to resolve the residual of the composite name and thus complete the requested operation.

The information returned by the GDA in the progress record is as follows:

Resolved part of name
: This consists of the part of the name that was successfully resolved, and information about it.

  — global root UUID; this identifier has the architected value of:

  ```
  11ed2286-49bb-11ca-8f29-08002b0dc46c
  ```

  — the cell name (global compound name, including the **/...** prefix)

  — length of the cell name.

Unresolved part of name
: This consists of the part of the name that was not successfully resolved, and information about it.

  — target cell root UUID

  — residual of the name (may be empty, with no leading / (slash) character)

  — length of the residual.

4. A GDA operation is considered to be successful only if the entity addressed by the global compound name represents a cell entry in the global name space, containing information about a cell.

Replica sets

This consists of location information for clearinghouses in the targeted CDS server (a *clearinghouse* is a database containing physical instances of CDS directories). If multiple replicated clearinghouses of the targeted cell are catalogued in the global name space entry, multiple set members are returned.

— clearinghouse types (*GDA*, *Master*, or *Read-only*)

— clearinghouse UUID

— clearinghouse name

— set of protocol towers (see the **DCE Remote Procedure Call** specification for details).

2.2.1 Hierarchical Cells

The Global Directory Agent (GDA) is also the gateway between cells in a cell hierarchy. In a hierarchical cell configuration, *child cells* are registered in the cell name space of a *parent cell*. The top-level parent in a cell hierarchy must be registered in a global name space. The parent cell's CDS acts as a higher-level directory service for the child cells.

A child cell's fully-qualified global name consists of the global name of the parent cell with the CDS name of the child cell appended to it. The GDA parses the name according to the syntax rules for DCE names; once it identifies the name of the parent cell it issues a request to that cell's CDS. The progress record returned by a successful GDA request corresponds to the progress record described in Section 2.2 on page 21.

2.3 Cell Registration in Global Name Space

The location information for cells is stored in separate global name space entries in accordance with the underlying rules of the global directory services. These entries need not be leaf nodes of the particular directory service.

The cell entries in the global name space uniquely identify the cell itself, as well as the location and type of CDS instances. If multiple instances of a CDS are available, the cell entry catalogues one record for each clearinghouse. The information that is required to catalogue a cell is defined as follows:

- the universal unique identifier (UUID) of the cell root directory, in hexadecimal notation (see the **DCE Remote Procedure Call** specification)
- the name of the clearinghouse, which may be a global or cell relative composite name
- the UUID of the clearinghouse, in hexadecimal notation

  This is the unique identifier of the clearinghouse in the target cell; the binding to the cell name may not be sufficient to reach the cell, because the name of the clearinghouse may change.
- the type of the clearinghouse (whether it is writable (*master*) or not (*read-only*))
- the network address of the clearinghouse.

2.3.1 Cell Registration in X.500

Cell information is contained in two X.500 attribute types, **CDS-Cell** and **CDS-Replica**. These attributes are either added to an existing directory entry or created along with a new specific directory entry. The syntax for both **CDS-Cell** and **CDS-Replica** attribute values is an octet string (see Appendix G). A single attribute value is formed as a sequence of individual fields, as specified below.

CDS-Cell contains the non-recurring cell information, as defined in Table 2-1.

Contents	**Syntax**	**Presence**
Target name space UUID	Printable string terminated by \0	Mandatory
Cell root directory UUID	Printable string terminated by \0	Only a \0 char if empty
Cell root directory name	Printable string terminated by \0	Only a \0 char if empty

Table 2-1 CDS-Cell Attribute format

CDS-Replica contains the set of recurring cell information, as defined in Table 2-2.

Contents	**Syntax**	**Presence**
Clearinghouse type	1 octet	**G**DA, **M**aster or **R**ead-only
Clearinghouse UUID	Printable string terminated by \0	Mandatory
Clearinghouse name	Printable string terminated by \0	Mandatory
Length of tower string	4 octets numeric string	Value from 1-325
Tower set	Octet string (with \0 separated set members)	Length from previous field

Table 2-2 CDS-Replica Attribute format

The UUIDs are represented in string form, as specified in the **DCE Remote Procedure Call** specification.

The tower set contains network address information, consisting of a set of null-terminated RPC string binding representations of protocol towers (see the **DCE Remote Procedure Call** specification).

For both attribute types, the attribute values (represented by octet strings) will vary in length. The current attribute value length is automatically stored in the X.500 directory system. The maximum attribute lengths must be defined in the X.500 schema (see Chapter 8).

2.3.2 Cell Registration in DNS

Cell information is contained in a pair of *resource records* (see RFC 1033 for recommendations on the contents of resource records). The resource records can be added to existing DNS nodes, or stored in newly created Internet domain names. The NIC Registrar is responsible for registering a domain name.

The resource records have the following type and contents:

- The type of the first record is **AFSDB** (AFS Data Base) or **MX**.

  The **ASFDB** or **MX** record indicates the host system on which the CDS server (replica, subtype 2) of the named cell exists. The DNS server also returns the associated Internet address of this host as additional data with the reply containing the **AFSDB** or **MX** record. The protocol class of this resource record is **IN** (Internet). **TTL** (time-to-live) is a value, expressed in seconds, after the elapse of which the data will no longer be considered valid in a DNS cache (the default value is set to the equivalent of one week).

- The type of the second record is **TXT** (text).

  The **TXT** record contains the set of recurring cell information, as well as the clearinghouse type and its address information. The fields in a **TXT** record are separated by white space, and their contents are case-insensitive.

  The first **TXT** field indicates the version (1). The next field contains the UUID of the cell root directory. Following this is a three-field group defining the clearinghouse managed by the server:

  — The first field of the triplet contains the type of the clearinghouse, and must contain one of the keywords **GDA**, **Master** or **Readonly**. Only the first character of the keyword is significant.

  — The second field contains the name of the clearinghouse. This name is the canonical string representation of the fully qualified global name.

  — The final field in the triplet contains the clearinghouse UUID.

  The hostname of the clearinghouse is also included in the **TXT** record, to enable the corresponding **AFSDB** and **TXT** records to be matched.

Additional pairs of **AFSDB** and **TXT** records are used to define multiple clearinghouses.

CAE Specification

Part 2

Application Programming Interfaces

The Open Group

Chapter 3

X/Open Directory Service API (XDS)

This chapter identifies the conformance requirements of the X/Open Directory Services (XDS) API implementations for the DCE. For the full X/Open XDS API, see the referenced **XDS** specification.

The XDS interface comprises a number of functions, together with OM classes of OM objects, which are used as the arguments and results of the functions. Both the functions and the OM objects are based on the Abstract Service specified in the ISO 9594-3 standard.

The interface models the directory interactions as service requests made through a number of interface functions, which take a number of input arguments. Each valid request causes an operation to be performed by the directory service, which eventually returns a status and any result of the operation.

All interactions between the user and the directory service belong to a *session*, which is represented by an OM object passed as the first argument to most interface functions.

The other arguments to the functions include a context and various service-specific arguments.

3.1 XDS Conformance

The XDS interface defines an API that application programs can use to access the functionality of the underlying directory service. The DCE XDS API conforms to the **XDS** specification. Both GDS and CDS are supported by the DCE XDS API.

DCE XDS is characterised by the following:

- It supports a synchronous interface. Asynchronous operations within the same thread are not supported.
- It supports all synchronous interface functions. The two asynchronous-specific functions are handled as follows:

  ds_abandon( )
  : Calling this routine does not cause a directory service abandon operation to be performed; instead, the routine returns a [DS_C_ABANDON_FAILED] ([DS_E_TOO_LATE]) error.

  ds_receive_result( )
  : If there are any outstanding operations (when multiple threads issue XDS calls in parallel), this functions returns [DS_SUCCESS], with the *completion_flag_return* argument set to a value of DS_OUTSTANDING_OPERATIONS. If no XDS calls are outstanding, then this function returns [DS_SUCCESS], and the *completion_flag_return* argument is set to a value of DS_NO_OUTSTANDING_OPERATION.
- Automatic connection management is not provided. The *ds_bind*( ) and *ds_unbind*( ) functions always try to set up or release directory service connections immediately.
- Support for local strings. XDS supports the mapping to / from local string formats. String mapping is supported by the following XDS / XOM functions:

  — *dsX_extract_attr_values*( )

  — *omX_object_to_string*( )

  — *omX_string_to_object*( )

  — *om_get*( )

  — *om_read*( )

  String mapping is requested by setting the *local_strings* boolean parameter to OM_TRUE. Currently supported mappings are:

  — T.61 String to / from ISO 8859-1 (Latin-1)

DCE XDS supports five packages, one of which is mandatory and the other four of which are optional. Use of the optional packages is negotiated through the *ds_version*( ) routine. The packages are as follows:

- The Directory Service Package (as defined in the **XDS** specification) is mandatory. This also includes the directory service errors.
- The Basic Directory Contents Package (as defined in the **XDS** specification) is optional.
- The Strong Authentication Package (as defined in the **XDS** specification) is optional.
- The MHS Directory User Package (as defined in the **XDS** specification) is optional.
- The Global Directory Service Package (as defined in Section 3.7 on page 35) is optional.

None of the OM classes defined in these packages are encodable. As a result, DCE XDS application programmers do not require the use of the XOM functions *om_encode*( ) and *om_decode*( ), which are not supported by the DCE XOM API.

3.2 XDS Functions

The OSI Directory Service standards define Abstract Services that requestors use to interact with the directory. Each of these Abstract Services maps to a single function call, and the detailed specifications of the calls are given in the XDS reference pages in the **XDS** specification. The services, and the function calls to which they map, are as follows:

DirectoryBind
Maps to *ds_bind*( ).

DirectoryUnbind
Maps to *ds_unbind*( ).

Read
Maps to *ds_read*( ).

Compare
Maps to *ds_compare*( ).

Abandon
Maps to *ds_abandon*( ).

List
Maps to *ds_list*( ).

Search
Maps to *ds_search*( ).

AddEntry
Maps to *ds_add_entry*( ).

RemoveEntry
Maps to *ds_remove_entry*( ).

ModifyEntry
Maps to *ds_modify_entry*( ).

ModifyRDN
Maps to *ds_modify_rdn*( ).

The XDS function *ds_receive_result*( ), which is used with asynchronous operations, has no counterpart in the Abstract Service.

The *ds_initialize*( ), *ds_shutdown*( ) and *ds_version*( ) functions are used to control the XDS API, and do not initiate any directory operations.

The interface functions are summarised in Table 3-1 on page 30.

Function	**Description**
ds_abandon( )	Abandons the result of a pending asynchronous operation. This function is not supported.
ds_add_entry( )	Adds a leaf entry to the DIT (Directory Information Tree).
ds_bind( )	Opens a session with a Directory User Agent (DUA), which in turn connects to a Directory Service Agent (DSA).
ds_compare( )	Compares a purported attribute value with the attribute value stored in the Directory Information Base (DIB) for a particular entry.
ds_initialize( )	Initialises the XDS interface.
ds_list( )	Enumerates the names of the immediate subordinates of a particular directory entry.
ds_modify_entry( )	Atomically performs modification of a directory entry.
ds_modify_rdn( )	Changes the Relative Distinguished Name (RDN) of a leaf entry.
ds_read( )	Queries information on a particular directory entry by name.
ds_receive_result( )	Retrieves the result of an asynchronously executed function. This function is not supported.
ds_remove_entry( )	Removes a leaf entry from the DIT.
ds_search( )	Finds entries of interest in a portion of the directory information tree.
ds_shutdown( )	Discards a workspace.
ds_unbind( )	Unbinds from a directory session.
ds_version( )	Negotiates features of the interface and service.

Table 3-1 XDS Interface Functions

The XDS interface functions are fully specified in the **XDS** specification. The following sections specify the extensions made for the DCE XDS API for each of the interface functions.

3.2.1 ds_abandon( )

The DCE XDS interface does not support asynchronous operations. Thus, *ds_abandon*( ) and *ds_receive_result*( ) are redundant. A *ds_abandon*( ) call always returns a [DS_C_ABANDON_FAILED] ([DS_E_TOO_LATE]) error.

3.2.2 ds_add_entry( )

Note the following in regard to the *ds_add_entry*( ) operation:

- Only leaf objects (that is, objects that are not CDS directory objects) can be added to CDS through the XDS interface.
- Only the **DS_A_COMMON_NAME** and **DS_A_MEMBER** attributes are valid for the **DS_O_GROUP_OF_NAMES** object in CDS.
- GDS-structured attribute types are not supported by CDS. If an attempt is made to add a GDS-structured attribute type to CDS, *ds_add_entry*( ) returns a [DS_C_ATTRIBUTE_ERROR] ([DS_E_CONSTRAINT_VIOLATION]) error.

Since CDS does not support the X.500 schema rules, some CDS objects may not contain GDS-mandatory attributes, such as object class and so on.

3.2.3 ds_bind( )

In order to use CDS when GDS is not active, *ds_bind*( ) must be called with the value of the *session* argument set to DS_DEFAULT_SESSION.

3.2.4 ds_compare( )

Note the following in regard to the *ds_compare*( ) operation:

- In CDS, the naming attribute of an object is not stored in the object's attribute list. Thus, in CDS, an attempted *ds_compare*( ) of a naming attribute value with the naming attribute value of the directory object always fails to yield a match.
- GDS-structured types are not supported by CDS. If a GDS-structured attribute type is used as an argument to a *ds_compare*( ) on a CDS object, the call returns the error [DS_C_ATTRIBUTE_ERROR] ([DS_E_CONSTRAINT_VIOLATION]).
- In CDS, *ds_compare*( ) can only be used on leaf objects; if it is called with a non-leaf object, it returns a [DS_C_NAME_ERROR] ([DS_E_NO_SUCH_OBJECT]) error.

3.2.5 ds_initialize( )

No modifications apply.

3.2.6 ds_list( )

For CDS, enumeration can be performed only on directories (that is, entries that are not leaf objects); if enumeration of a leaf object is attempted, the call returns a [DS_C_NAME_ERROR] ([DS_E_NO_SUCH_OBJECT]) error.

3.2.7 ds_modify_entry( )

Note the following in regard to the *ds_modify_entry*( ) operation:

- X.500 naming schema rules do not apply in CDS. Thus, the following attribute errors are never returned from CDS operations:

  — [DS_E_NO_SUCH_ATTRIBUTE_OR_VALUE]

  — [DS_E_ATTRIBUTE_OR_VALUE_EXISTS].
- Naming operations that would normally return these errors in GDS succeed when executed in CDS. In particular, attempting to add an attribute that already exists does not cause an error with a CDS object. Instead, the values of the attribute to be added are combined with the values of the existing attribute.
- GDS-structured attribute types are not supported by CDS. If a GDS-structured attribute type is used as an argument to a *ds_modify_entry*( ) call on a CDS object, the routine returns a [DS_C_ATTRIBUTE_ERROR] ([DS_E_CONSTRAINT_VIOLATION]) error.
- In CDS, *ds_modify_entry*( ) can be used only on leaf objects; otherwise, it returns a [DS_C_NAME_ERROR] ([DS_E_NO_SUCH_OBJECT]) error.

3.2.8 ds_modify_rdn( )

CDS does not support the *ds_modify_rdn*( ) operation. Attempting this operation on a CDS object results in an error return of [DS_C_SERVICE_ERROR] ([DS_E_UNWILLING_TO_PERFORM]).

3.2.9 ds_read( )

Note the following in regard to the **ds_read**( ) operation:

- Since CDS does not support the X.500 schema rules, some CDS objects may not contain GDS-mandatory attributes, such as object class, and so on. In CDS, an attempted read of a CDS alias object fails if the **DS_A_ALIASED_OBJECT_NAME** does not exist in the object. Instead, *ds_read*( ) returns a [DS_C_NAME_ERROR] ([DS_E_NO_SUCH_OBJECT]) error.
- In CDS, the naming attribute of an object is not stored in the object's attribute list. Thus, a *ds_read*( ) of a CDS object does not return this attribute in the object's attribute list.

3.2.10 ds_receive_result( )

The DCE XDS interface does not support asynchronous operations. Thus, *ds_abandon*( ) and *ds_receive_result*( ) are redundant. A *ds_receive_result*( ) function call always returns with *DS_status* set to [DS_SUCCESS], and the *completion_flag_return* argument set to DS_NO_OUTSTANDING_OPERATION.

3.2.11 ds_remove_entry( )

No modifications apply.

3.2.12 ds_search( )

CDS does not support the *ds_search*( ) operation. An attempt to perform a *ds_search*( ) in CDS results in an error of [DS_C_SERVICE_ERROR] ([DS_E_UNWILLING_TO PERFORM]) being returned.

3.2.13 ds_shutdown( )

No modifications apply.

3.2.14 ds_unbind( )

No modifications apply.

3.2.15 ds_version( )

No modifications apply.

3.3 XDS Function Call Results

All XDS functions, with the exception of *ds_initialize( )*, return a value of type *DS_status*, which is the C function result of the call. If the function is successful, then *DS_status* returns with a value of [DS_SUCCESS]. If the function does not complete successfully, then *DS_status* takes either the error [DS_NO_WORKSPACE] or a private error object.

Most XDS functions also return data in an *invoke_id_return* argument, which identifies the particular invocation; and each of the interrogation operations returns data in a *result_return* argument. The *invoke_id_return* and *result_return* values are returned using pointers that are supplied as arguments of the C function.

These three types of function results are described in the **XDS** specification.

3.3.1 invoke_id_return Parameter

All interface functions that invoke a directory service operation return an *invoke_id_return* argument, which is an integer that identifies the particular invocation of an operation. This information is meaningful only if asynchronous operations are supported. Since DCE XDS does not support asynchronous operations, the *invoke_id_return* value is meaningless in DCE XDS.

The affected interface functions are:

- *ds_add_entry*( )
- *ds_compare*( )
- *ds_list*( )
- *ds_modify_entry*( )
- *ds_modify_rdn*( )
- *ds_read*( )
- *ds_remove_entry*( )
- *ds_search*( ).

DCE application programmers must still supply this argument, as described in the XDS reference pages (see the **XDS** specification), but the value returned therein should be ignored.

3.4 Synchronous Operations

Since asynchronous use of the interface within the same thread is not supported, the value of the **DS_ASYNCHRONOUS** OM attribute in **DS_C_CONTEXT** is always OM_FALSE, causing all operations within the same thread to be synchronous.

In synchronous mode, all functions wait until the operation is complete before returning. The thread of control is blocked within the interface during the time which elapses between the calling of a function and its return, and the function result can be used immediately after the function returns.

Implementations may define a limit on the number of asynchronous operations that can be outstanding at any one time in any one session. The limit is defined by the implementation-defined constant DS_MAX_OUTSTANDING_OPERATIONS. In DCE XDS this constant always has the value 0 (zero), because asynchronous operations are not supported.

All errors occurring during a synchronous request are reported when the function returns.

The **DS_FILE_DESCRIPTOR** OM attribute of **DS_C_SESSION** is not used by the DCE XDS API. It is always set to DS_NO_VALID_FILE_DESCRIPTOR.

3.5 Security and XDS

The **XDS** specification does not define a security interface in order to avoid possibly constraining the security features of existing directory implementations.

DCE GDS proves an extension to the XDS API for security support. This is achieved at the XDS API level through a new **DSX_C_GDS_SESSION** session object which contains information on the security mechanism that should be used. Simple authentication through the use of name and password, and external authentication based on DCE security are supported. (See Section 3.7 on page 35 for additional information.)

3.6 Automatic Connection Management

A directory service implementation can provide automatic management of the association or connection between the user and the directory service, making and releasing connections at its discretion.

DCE XDS does not support automatic connection management. A DSA connection is established when *ds_bind*( ) is successfully called, and released when *ds_unbind*( ) is successfully called.

3.7 Global Directory Service Package

The Global Directory Service Package (GDSP) is an OSF extension to the XDS interface. Applications must negotiate use of this package, by calling *ds_version*( ), before using any of the package's features. If an application attempts to use features specific to this package without first negotiating its use, an error (for example, [OM_NO_SUCH_CLASS]) is returned by the DCE XOM function it attempted to execute.

The object identifier associated with the GDSP is:

```
{iso(1) identified-organisation(3) icd-ecma(0012) member-company(2)\
 siemens-units(1107) sni(1) directory(3) xds-api(100) gdsp(0)}
```

It has the following encoding:

```
\x2B\xC\x2\x88\x53\x1\x3\x64\x0
```

The GDSP object identifier is represented by the constant DSX_GDS_PKG. This constant, together with the other C constants associated with the package, are contained in the **<xdsgds.h>** header file (see Appendix H).

In the following sections, the GDSP's attribute types are introduced first; descriptions of its object classes follow. Finally, the OM class hierarchy and OM class definitions required to support the new attribute types are described.

3.7.1 GDSP Attribute Types

Additional directory attribute types are used with the GDS package. Each attribute type has its own object identifier, held as the value of the OM attribute **DS_ATTRIBUTE_TYPE**. These object identifiers are represented in the interface by constants with the same name as the directory attribute they identify, prefixed by DSX_A_.

Table 3-2 on page 36 shows the names of the GDSP attribute types, together with the BER encoding of the object identifiers associated with each of them. The third column of the table shows the hexadecimal values of the octets of the BER encoding of the object identifier in hexadecimal representation. All these object identifiers are derived from the root:

```
{iso(1) identified-organisation(3) icd-ecma(0012) member-company(2)
 siemens-units(1107) sni(1) directory(3) attribute-type(4)}
```

Package	Attribute Type	Object Identifier BER (Hexadecimal Value)
GDSP	**DSX_A_ACL**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x01
GDSP	**DSX_A_AT**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x06
GDSP	**DSX_A_CDS_CELL**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x0D
GDSP	**DSX_A_CDS_REPLICA**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x0E
GDSP	**DSX_A_CLIENT**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x0A
GDSP	**DSX_A_DEFAULT_DSA**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x08
GDSP	**DSX_A_DNLIST**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x0B
GDSP	**DSX_A_LOCAL_DSA**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x09
GDSP	**DSX_A_MASTER_KNOWLEDGE**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x00
GDSP	**DSX_A_OCT**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x05
GDSP	**DSX_A_SHADOWED_BY**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x03
GDSP	**DSX_A_SHADOWING_JOB**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x0C
GDSP	**DSX_A_SRT**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x04
GDSP	**DSX_A_TIME_STAMP**	\x2B\x0C\x02\x88\x53\x01\x03\x04\x02

Table 3-2 Object Identifiers for GDSP Attribute Types

Table 3-3 shows the names of the attribute types, together with the OM value syntax used in the interface to represent each attribute's values. The table also indicates: the range of lengths in octets permitted for the string types; whether the attribute can be multi-valued; and which matching rules are provided for the syntax.

Attribute Type	OM Value Syntax	Value Length	Multi-Valued	Matching Rules
DSX_A_ACL	Object (**DSX_C_GDS_ACL**)	—	no	E
DSX_A_AT	String (**OM_S_PRINTABLE_STRING**)	1-101	yes	E,S
DSX_A_CDS_CELL	String (**OM_S_OCTET_STRING**)	1-284	no	E
DSX_A_CDS_REPLICA	String (**OM_S_OCTET_STRING**)	1-905	yes	E
DSX_A_CLIENT	Only a cache attribute	—	—	—
DSX_A_DEFAULT_DSA	Only a cache attribute	—	—	—
DSX_A_DNLIST	Object (**DS_C_DS_DN**)	—	yes	E,S
DSX_A_LOCAL_DSA	Only a cache attribute	—	—	—
DSX_A_MASTER_KNOWLEDGE	Object (**DS_C_DS_DN**)	—	no	E,S
DSX_A_OCT	String (**OM_S_PRINTABLE_STRING**)	1-397	yes	E,S
DSX_A_SHADOWED_BY	Not used yet	—	—	—
DSX_A_SHADOWING_JOB	Not used yet	—	—	—
DSX_A_SRT	String (**OM_S_PRINTABLE_STRING**)	1-29	yes	E,S
DSX_A_TIME_STAMP	String (**OM_S_UTC_TIME_STRING**)	11-17	no	E,O

Table 3-3 Values for GDSP Attribute Types

In the **Matching Rules** column, the abbreviations have the following meanings:

E	The matching rule determines whether two values are equal.
S	The matching rule identifies one value as a substring of the other.
O	The matching rule determines the ordering of two values.

Descriptions of GDSP Attribute Types

See the **XDS** specification for information on general matching rules.

DSX_A_ACL
The contents of this attribute describe the access rights for one or more directory service users.

DSX_A_AT
The contents of this attribute describe the attribute types permitted in GDS.

DSX_A_CDS_CELL and **DSX_A_CDS_REPLICA**
The contents of these attributes consist of the information necessary for contacting a remote DCE cell. These two attributes always exist together in the same object. See Section 2.3 on page 23 for its representation.

DSX_A_CLIENT
This attribute applies only to the cache. It identifies the entry which holds the DUA's Presentation Address. Its OM syntax is **OM_S_PRINTABLE_STRING** and its value is CLIENT.

DSX_A_DEFAULT_DSA
This attribute applies only to the cache. It identifies an entry which holds the Distinguished Name (DN) of the DUA's default DSA. Its OM syntax is **OM_S_PRINTABLE_STRING** and its value is DEFAULT-DSA.

DSX_A_DNLIST
The contents of this attribute are used internally by the GDS DSA.

DSX_A_LOCAL_DSA
This attribute applies only to the cache. It identifies an entry which holds the Distinguished Name (DN) of the DUA's local DSA. Its OM syntax is **OM_S_PRINTABLE_STRING** and its value is LOCAL-DSA.

DSX_A_MASTER_KNOWLEDGE
The contents of this attribute consist of the Distinguished Name (DN) of the DSA that holds the master copy of this entry.

DSX_A_OCT
The contents of this attribute consist of a description of the object classes supported by the DSA.

DSX_A_SHADOWED_BY and **DSX_A_SHADOWING_JOB**
These two GDSP attributes are intended for future use.

DSX_A_SRT
The contents of this attribute consist of a description of the structure of the DNs (Distinguished Names) permitted in GDS.

DSX_A_TIME_STAMP
This attribute is part of the **DSX_O_SCHEMA** object. It contains the creation time of the **DSX_O_SCHEMA** object.

3.7.2 GDSP Object Classes

The only object class specific to the GDSP is **DSX_O_SCHEMA** (see Table 3-4). It is stored in GDS as an object directly under the directory root. The most important attributes of the **DSX_O_SCHEMA** object are the three recurring ones **DSX_A_OCT**, **DSX_A_AT** and **DSX_A_SRT**, which describe the GDS directory information tree (DIT) structure.

The third column of the table shows the hexadecimal values of the octets of the BER encoding of the object identifier in hexadecimal representation. This object identifier is derived from the root:

```
{iso(1) identified-organisation(3) icd-ecma(0012) member-company(2)
 siemens-units(1107) sni(1) directory(3) object-class(6)}
```

Package	Attribute Type	Object Identifier BER (Hexadecimal Value)
GDSP	**DSX_O_SCHEMA**	\x2B\x0C\x02\x88\x53\x01\x03\x06\x00

Table 3-4 Object Identifier for GDSP Object Classes

3.7.3 GDS OM Class Hierarchy

The additional OM classes used by the GDS package are organised hierarchically. In the following list, subclassification is indicated by indentation; it shows which classes inherit additional OM attributes from their OM superclasses.

- **OM_C_OBJECT** (defined in the OM package)
    - **DS_C_SESSION** (defined in the Directory Service Package)
        - **DSX_C_GDS_SESSION**
    - **DS_C_CONTEXT** (defined in the Directory Service Package)
        - **DSX_C_GDS_CONTEXT**
    - **DSX_C_GDS_ACL**
    - **DSX_C_GDS_ACL_ITEM**

None of the OM classes in the preceding list are encodable using *om_encode*( ) and *om_decode*( ).

DSX_C_GDS_ACL

An instance of OM class **DSX_C_GDS_ACL** describes up to five categories of rights for one or more directory users.

An instance of this OM class has the OM attributes of its superclass, **OM_C_OBJECT**, in addition to the OM attributes listed in Table 3-5.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
DSX_MODIFY_PUBLIC	Object (**DSX_C_GDS_ACL_ITEM**)	—	0-4	—
DSX_READ_STANDARD	Object (**DSX_C_GDS_ACL_ITEM**)	—	0-4	—
DSX_MODIFY_STANDARD	Object (**DSX_C_GDS_ACL_ITEM**)	—	0-4	—
DSX_READ_SENSITIVE	Object (**DSX_C_GDS_ACL_ITEM**)	—	0-4	—
DSX_MODIFY_SENSITIVE	Object (**DSX_C_GDS_ACL_ITEM**)	—	0-4	—

Table 3-5 OM Attributes of DSX_C_GDS_ACL

The OM attributes of **DSX_C_GDS_ACL** are as follows:

DSX_MODIFY_PUBLIC
This attribute specifies the user, or subtree of users, that can modify attributes classified as public attributes.

DSX_READ_STANDARD
This attribute specifies the user, or subtree of users, that can read attributes classified as standard attributes.

DSX_MODIFY_STANDARD
This attribute specifies the user, or subtree of users, that can modify attributes classified as standard attributes.

DSX_READ_SENSITIVE
This attribute specifies the user, or subtree of users, that can read attributes classified as sensitive attributes.

DSX_MODIFY_SENSITIVE
This attribute specifies the user, or subtree of users, that can modify attributes classified as sensitive attributes.

DSX_C_GDS_ACL_ITEM

An instance of OM class **DSX_C_GDS_ACL_ITEM** is a component of an instance of OM class **DSX_C_GDS_ACL**. It specifies the user, or subtree of users, to whom an access right applies.

An instance of this OM class has the OM attributes of its superclass, **OM_C_OBJECT**, in addition to the OM attributes listed in Table 3-6.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
DSX_INTERPRETATION	Enum (**DSX_Interpretation**)	—	1	—
DSX_USER	Object(**DS_C_DS_DN**)	—	1	—

Table 3-6 OM Attributes of DSX_C_GDS_ACL_ITEM

The OM attributes of a **DSX_C_GDS_ACL_ITEM** are as follows:

DSX_INTERPRETATION
This attribute specifies the scope of the access right. It can have one of the following values:

— **DSX_SINGLE_OBJECT** indicates that the access right is granted to the user specified in the **DSX_USER** OM attribute.

— **DSX_ROOT_OF_SUBTREE** indicates that the access right is granted to all users in the subtree below the name specified in the **DSX_USER** OM attribute.

DSX_USER
This attribute contains the Distinguished Name of the user, or subtree of users, to whom an access right applies.

DSX_C_GDS_CONTEXT

An instance of OM class **DSX_C_GDS_CONTEXT** comprises per-operation arguments that are accepted by most of the interface functions. The GDS package supports additional service controls that are defined by the **DSX_C_GDS_CONTEXT** OM class.

An instance of this OM class has the OM attributes of its superclasses, **OM_C_OBJECT** and **DS_C_CONTEXT**, in addition to the OM attributes listed in Table 3-7.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Service Controls				
DSX_DUAFIRST	**OM_S_BOOLEAN**	—	1	OM_FALSE
DSX_DONT_STORE	**OM_S_BOOLEAN**	—	1	OM_TRUE
DSX_NORMAL_CLASS	**OM_S_BOOLEAN**	—	1	OM_FALSE
DSX_PRIV_CLASS	**OM_S_BOOLEAN**	—	1	OM_FALSE
DSX_RESIDENT_CLASS	**OM_S_BOOLEAN**	—	1	OM_FALSE
DSX_USEDSA	**OM_S_BOOLEAN**	—	1	OM_TRUE
DSX_DUA_CACHE	**OM_S_BOOLEAN**	—	1	OM_FALSE
DSX_PREFER_ADM_FUNCS	**OM_S_BOOLEAN**	—	1	OM_FALSE
DSX_SIGN_MECHANISM	Enum(**DSX_Sign_Mechanism**)	—	0-1	—
DSX_PROT_REQUEST	Enum(**DSX_Prot_Request**)	—	0-1	—

Table 3-7 OM Attributes of DSX_C_GDS_CONTEXT

The OM attributes of the **DSX_C_GDS_CONTEXT** OM class are as follows:

DSX_DUAFIRST
The value of this attribute defines whether the DUA cache or the DSA needs to be read first for query operations. The default value is OM_FALSE; that is, search the DSA first, if not found then search the DUA cache.

DSX_DONT_STORE
The value of this attribute specifies whether the information read from the DSAs by the query functions also needs to be stored in the DUA cache. When this service control is set to OM_TRUE (the default value), nothing is stored in the DUA cache.

When the value of this attribute is set to OM_FALSE, the information read is stored in the DUA cache, and the objects returned by *ds_list*( ) and *ds_compare*( ) are stored in the cache without their associated attribute information. The objects returned by *ds_read*( ) and *ds_search*( ) are stored in the cache with all their attributes that are capable of being stored in the cache; these are public attributes, except for the ACL attribute. This information is stored in the cache only when a list of requested attributes is supplied. If all attributes are requested, then nothing is stored in the cache.

The DUA cache categorises the information stored into three different memory classes. The user specifies the category with the following service control attributes:

DSX_NORMAL_CLASS
If this attribute is set to OM_TRUE, the entry in the DUA cache is assigned to the class of normal objects. When the number of entries in this class exceeds a maximum value, the entry that has not been addressed for the longest period of time is removed from the DUA cache.

DSX_PRIV_CLASS
If this attribute is set to OM_TRUE, the entry in the DUA cache is assigned to the class of

privileged objects. Such entries can be removed from the class in the same way as normal objects. By using this memory sparingly, the user can protect entries from deletion.

DSX_RESIDENT_CLASS
If this attribute is set to OM_TRUE, the entry in the DUA cache is assigned to the class of resident objects. An entry in this memory class is never removed automatically; it can only be removed by a call to *ds_remove_entry*( ). The number of entries is limited; if this limit is exceeded, *ds_add_entry*( ) reports an error.

Note: Only one of the above service control attributes can be OM_TRUE at one time. Also, the *ds_add_entry*( ) function also evaluates these service control bits if it is used on the DUA cache.

DSX_DUA_CACHE and **DSX_USEDSA**
These attributes define whether the entries in the DUA cache or in the DSA, or both, need to be used when providing the service specified in the operation. Depending on the values of these attributes, the following situations can arise:

— If **DSX_DUA_CACHE** and **DSX_USEDSA** are both OM_TRUE, the *ds_add_entry*( ) and *ds_remove_entry*( ) functions report an error.

  The query functions evaluate the service controls **DS_DONT_USE_COPY** and **DSX_DUAFIRST**. If **DS_DONT_USE_COPY** is OM_FALSE, then the value of **DSX_DUAFIRST** determines whether the DUA cache or the DSA is read first. If **DS_DONT_USE_COPY** is OM_TRUE, information from the DSA only is read.

— If **DSX_DUA_CACHE** is OM_TRUE and **DSX_USEDSA** is OM_FALSE, the *ds_add_entry*( ) and *ds_remove_entry*( ) functions, as well as the query functions, go only to the DUA cache.

— If **DSX_DUA_CACHE** is OM_FALSE and **DSX_USEDSA** is OM_TRUE, the *ds_add_entry*( ) and *ds_remove_entry*( ) functions, as well as the query functions, go only to the DSA.

— If **DSX_DUA_CACHE** and **DSX_USEDSA** are both OM_FALSE, the *ds_add_entry*( ) and *ds_remove_entry*( ) functions, and the query functions, report an error.

All other functions always operate on the currently connected DSA.

DSX_PREFER_ADM_FUNCS
The value of this attribute specifies whether the three following optional attributes are returned in an operation:

— **DSX_A_MASTER_KNOWLEDGE**, which contains the Distinguished Name of the DSA that holds the master copy of an entry

— **DSX_A_ACL**, which is used for GDS access control

— **DS_A_USER_PASSWORD** is an attribute of the **DS_O_DSA** object class, which is used by the GDS shadowing mechanism.

The **DSX_A_MASTER_KNOWLEDGE** and **DSX_A_ACL** attributes are present in every GDS entry.

When an application requests that all attributes be returned in an operation, it can prevent any of the above three optional attributes from being returned by setting the value of **DSX_PREFER_ADM_FUNCS** to OM_FALSE.

If GDS applications (for example, GDS administration) require these attributes, they are obtained by setting this service control to OM_TRUE.

DSX_SIGN_MECHANISM
This attribute is reserved for future use.

DSX_PROT_REQUEST
This attribute is reserved for future use.

Applications can assume that an object of OM class **DSX_C_GDS_CONTEXT**, created with default values for all its OM attributes, works with all interface functions. Note that an application can supply the constant **DS_DEFAULT_CONTEXT** as the context argument to GDS functions instead of creating a **DSX_C_GDS_CONTEXT** OM object with all default values.

The default form of **DSX_C_GDS_CONTEXT** is described in Table 3-8.

OM Attribute	Default Value
Common Arguments	
DS_OPERATION_PROGRESS	DS_OPERATION_NOT_STARTED
DS_ALIASED_RDNS	0
Service Controls	
DS_CHAINING_PROHIB	OM_TRUE
DS_DONT_DEREFERENCE_ALIASES	OM_FALSE
DS_DONT_USE_COPY	OM_TRUE
DS_LOCAL_SCOPE	OM_FALSE
DS_PREFER_CHAINING	OM_FALSE
DS_PRIORITY	DS_MEDIUM
Local Controls	
DS_ASYNCHRONOUS	OM_FALSE
DS_AUTOMATIC_CONTINUATION	OM_TRUE
Private Extensions	
DSX_DUAFIRST	OM_FALSE
DSX_DONT_STORE	OM_TRUE
DSX_NORMAL_CLASS	OM_FALSE
DSX_PRIV_CLASS	OM_FALSE
DSX_RESIDENT_CLASS	OM_FALSE
DSX_USEDSA	OM_TRUE
DSX_DUA_CACHE	OM_FALSE
DSX_PREFER_ADM_FUNCS	OM_FALSE
DSX_SIGN_MECHANISM	Absent
DSX_PROT_REQUEST	Absent

Table 3-8 Default DSX_C_GDS_CONTEXT

DSX_C_GDS_SESSION

An instance of OM class **DSX_C_GDS_SESSION** identifies and describes a particular link from an application program to a GDS DSA. This additional OM class is necessary if the user or application wants to do either or both of the following:

- specify an authentication mechanism for an authenticated bind
- specify the GDS directory identifier.

DSX_C_GDS_SESSION can be passed as an argument to *ds_bind*( ).

An instance of this OM class has the OM attributes of its superclasses, **OM_C_OBJECT** and **DS_C_SESSION**, in addition to the OM attributes listed in Table 3-9.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
DSX_PASSWORD	String(**OM_S_OCTET_STRING**)	—	0 or 1	—
DSX_DIR_ID	**OM_S_INTEGER**	—	1	1
DSX_AUTH_MECHANISM	Enum(**DSX_Auth_Mechanism**)	—	0-1	—
DSX_AUTH_INFO	String(**OM_S_OCTET_STRING**)	—	0-1	—

Table 3-9 OM Attributes of DSX_C_GDS_SESSION

The OM attributes of **DSX_C_GDS_SESSION** are as follows:

DSX_PASSWORD
The contents of this attribute consist of the password for the user credentials.

DSX_DIR_ID
The contents of this attribute are an identifier for distinguishing between several configurations of the directory service within a GDS installation. The valid range for this value is from 1 to 20.

DSX_AUTH_MECHANISM
This attribute, if present, identifies the authentication mechanism that the application requests. If it is absent or has the value **DSX_NONE_AT_ALL**, then a *ds_bind( )* without credentials (anonymous bind) is requested. This attribute can take the following values:

— **DSX_NONE_AT_ALL** requests that no authentication mechanism be used.

— **DSX_DEFAULT** requests the default authentication mechanism, DCE authentication (**DSX_DCE_AUTH**). The value for **DSX_DEFAULT** can be modified through the **XDS_DEF_AUTH_MECH** environment variable. This variable is checked by XDS following a *ds_initialize( )* function call.

— **DSX_SIMPLE** requests simple authentication by using the **DS_REQUESTOR** and **DSX_PASSWORD** attributes of the **DSX_C_GDS_SESSION** object.

— **DSX_SIMPLE_PROT1** is reserved for future use.

— **DSX_SIMPLE_PROT2** is reserved for future use.

— **DSX_DCE_AUTH** requests the use of the DCE authentication mechanism.

— **DSX_STRONG** is reserved for future use.

If an authentication mechanism is selected that is not currently supported, *ds_bind( )* returns a [DS_E_NOT_SUPPORTED] error. If the selected authentication mechanism requires the user's credentials that cannot be assembled, then a [DS_E_NO_INFO] error is returned.

DSX_AUTH_INFO
This attribute is reserved for future use.

Applications can assume that an object of OM class **DSX_C_GDS_SESSION**, created with default values for all its OM attributes, works with all interface functions. Note that an application can supply the constant **DS_DEFAULT_SESSION** as the session argument to *ds_bind*( ) to create a default session (instead of creating a **DSX_C_GDS_SESSION** OM object with all the default values), having already negotiated the GDS package.

Table 3-10 describes **DS_DEFAULT_SESSION**.

OM Attribute	Default Value
DS_DSA_ADDRESS	Value obtained from the cache or absent
DS_DSA_NAME	Value obtained from the cache or absent
DS_FILE_DESCRIPTOR	DS_NO_VALID_FILE_DESCRIPTOR
DSX_DIR_ID	1
DSX_AUTH_MECHANISM	Absent
DSX_AUTH_INFO	Absent

Table 3-10 Default DSX_C_GDS_SESSION

Chapter 4

X/Open OSI Abstract Data Manipulation (XOM)

This chapter identifies the conformance requirements of the X/Open OSI-Abstract-Data Manipulation (XOM) API implementations for the DCE. For the full X/Open XOM API, see the referenced **XOM** specification.

The **XOM** specification is the specification of a general purpose API for use in conjunction with other application-specific APIs, such as XDS.

XOM consists of the creation, examination, modification and deletion of potentially complex information objects. It presents to the programmer a uniform model of information based on the concept of classes of similar information objects.

The information objects to which XOM applies are those that arise in OSI — that is, those that correspond to the types defined by, or by means of, Abstract Syntax Notation One (ASN.1). The XOM API comprises tools for manipulating ASN.1 objects.

4.1 XOM Conformance

DCE XOM conforms to the **XOM** specification.

DCE XOM has the following characteristics:

- Multiple workspaces for XDS objects are supported.
- The OM package is supported.
- The *om_encode*( ) and *om_decode*( ) functions are not supported. The transfer of objects between workspaces is not envisaged within the DCE environment, and the OM classes used by the DCE XDS/XOM API are not encodable.
- Translation to local character sets is supported.

4.2 XOM Functions

Table 4-1 lists the functions of the XOM C interface.

Function	**Description**
om_copy( )	Copies a private object.
om_copy_value( )	Copies a string between private objects.
om_create( )	Creates a private object.
om_decode( )	This function is not supported by the DCE XOM interface, and returns with an [OM_FUNCTION_DECLINED] error.
om_delete( )	Deletes a private or service-generated object.
om_encode( )	This function is not supported by the DCE XOM interface, and returns with an [OM_FUNCTION_DECLINED] error.
om_get( )	Gets copies of attribute values from a private object.
om_instance( )	Tests an object's class.
om_put( )	Puts attribute values into a private object.
om_read( )	Reads a segment of a string in a private object.
om_remove( )	Removes attribute values from a private object.
om_write( )	Writes a segment of a string into a private object.

Table 4-1 XOM Interface Functions

The service interface comprises a number of functions, whose purpose and range of capabilities are described in the following sections.

4.2.1 om_copy( )

Creates an independent copy of an existing private object and all its subobjects. The copy is placed in the original's workspace, or in another workspace specified by the XOM application.

4.2.2 om_copy_value( )

Replaces an existing attribute value, or inserts a new value, into a private object. The value is a copy of an existing attribute value found in another private object. Both values must be strings.

4.2.3 om_create( )

Creates a new private object that is an instance of a particular class. The object can be initialised with the attribute values specified as initial in its class definition.

The service does not permit the API user explicitly to create instances of all classes, but rather only those indicated by a package's definition as having this property.

4.2.4 om_delete( )

Deletes a service-generated public object, or makes a private object inaccessible.

4.2.5 om_get( )

Creates a new public object that is an exact but independent copy of an existing private object. The caller can request certain exclusions, each of which reduces the copy to some part of the original. Exclusions can be as follows:

- attributes of types other than those specified
- values at positions other than those specified within an attribute
- the values of multi-valued attributes
- copies of (but not handles for) subobjects
- all attribute values (with the result that only an attribute's presence is indicated by the return).

The caller can also request that values be converted from one syntax to another before they are returned (that is, copied).

4.2.6 om_instance( )

Determines whether an object is an instance of a particular class. An application can determine an object's immediate class simply by inspection. However, this function is useful because it can be used to find whether an object is an instance of a particular class even if the object is an instance of a subclass of that class.

4.2.7 om_put( )

Places or replaces in a specified private object copies of the attribute values of some other specified public or private object.

The source values can be inserted:

- before any existing values in the destination attribute
- before the value at a specified position in the destination attribute
- after any existing destination values.

Alternatively, the source values can be made to overwrite all existing destination values, or only the values at specified positions in the destination attribute.

4.2.8 om_read( )

Reads a segment of a value of an attribute of a private object. The value must be a string. The value can first be converted from one syntax to another. This function enables the caller to read an arbitrarily long value without requiring that the service place a copy of the entire value in memory.

4.2.9 om_remove( )

Removes and discards particular values of an attribute of a private object. The attribute itself is removed if no values exist.

4.2.10 om_write( )

Writes a segment of a value of an attribute to a private object. The value must be a string. The segment can first be converted from one syntax to another. The written segment becomes the value's last segment, since any elements beyond it are truncated as a result of the call. This function enables the caller to write an arbitrarily long value without having to place a copy of the entire value in memory.

The intention of the interface definition is that each function be atomic; that is, that it will either carry out its assigned task to completion and then report success, or that it fails to carry out even a part of the task, whereupon it will report an exception. However, the service does not guarantee that a task is always carried out in full.

Chapter 5

XDS/XOM Convenience Functions

This chapter describes functions which are available to XDS/XOM programmers to help simplify and speed up the development of XDS applications. The convenience functions target two main areas:

- Filling, comparing and extracting objects
- Converting objects to and from local string formats.

5.1 String Handling

The convenience functions provide the ability to specify OM objects in string format by means of abbreviations.

X.500 attribute types can be specified either as abbreviations or object identifier strings. The mapping of the attribute abbreviations and object identifier strings to BER encoded object identifiers and the associated attribute syntaxes is determined by the XOM object information module.

The convenience functions are able to handle strings with special syntax. The strings can be broadly classified into the following:

- strings representing GDS attribute information
- strings representing structured GDS attribute information
- strings representing a structured GDS attribute value
- strings representing a Distinguished Name (DN)
- strings representing expressions.

5.2 Convenience Functions

Table 5-1 lists the convenience functions interfaces.

Function	**Description**
dsX_extract_attr_values( )	Extracts attribute values from an object.
omX_extract( )	Creates an exact, independent copy of an existing subobject.
omX_fill( )	Initializes an **OM_descriptor** structure.
omX_fill_oid( )	Initializes an **OM_descriptor** with an object identifier value.
omX_object_to_string( )	Converts an **OM_object** to string format.
omX_string_to_object( )	Converts a string to a new private object.

Table 5-1 Convenience Functions

The purpose and range of capabilities of the convenience functions are described in the following sections.

5.2.1 dsX_extract_attr_values( )

Extracts the attribute values associated with the specified attribute type from an OM object. The OM object must be of type **DS_C_ATTRIBUTE_LIST** or **DS_C_ENTRY_INFO**. This function returns an object containing an array of OM descriptors.

5.2.2 omX_extract( )

Creates a new public object, that is an exact but independent copy of an existing subobject in a private object. It is similiar to the *om_get( )* function but includes an additional parameter, *navigation_path*, which contains directions to the required object to be extracted. The client can request certain exclusions, each of which reduces the copy to a part of the original.

5.2.3 omX_fill( )

Initializes an **OM_descriptor** structure with user-supplied values for the structure's *type, syntax*, and value.

5.2.4 omX_fill_oid( )

Initializes an **OM_descriptor** structure with user-supplied values for the structures *type* and value. The syntax of the **OM_descriptor** is always set to **OM_S_OBJECT_IDENTIFIER_STRING**.

5.2.5 omX_object_to_string( )

Converts an object into a string format. The object can either be a client-generated or service-generated public or private object.

5.2.6 omX_string_to_object( )

Creates a new private object, which is built from the *string* and *class* input parameters.

Chapter 6

Name Service Independent Interface (NSI)

This chapter describes the Name Service Independent Interface (NSI), which is the low-level RPC-based naming interface that is fully specified in the **DCE Remote Procedure Call** specification. This interface utilises the Cell Directory Service (CDS) specified in Part 4.

6.1 Interface Functions

The tables in this section summarise the NSI interface functions.

Function	Description
rpc_ns_binding_export( )	Exports server binding information to a name service entry.
rpc_ns_binding_import_begin( )	Creates an import context for importing bindings from a name service.
rpc_ns_binding_import_done( )	Deletes a name service import context.
rpc_ns_binding_import_next( )	Returns a binding handle for a compatible server from a name service.
rpc_ns_binding_inq_entry_name( )	Returns the name of the entry in the name service database from which the binding information referenced by a server binding handle came.
rpc_ns_binding_lookup_begin( )	Creates a look-up context for importing bindings from a name service.
rpc_ns_binding_lookup_done( )	Deletes a name service look-up context.
rpc_ns_binding_lookup_next( )	Returns a vector of compatible binding handles from a name service.
rpc_ns_binding_select( )	Returns a binding handle from a vector of compatible server binding handles.
rpc_ns_binding_unexport( )	Removes binding information from an entry in a name service database.
rpc_ns_import_ctx_add_eval( )	Adds an evaluation routine to an import context.

Table 6-1 NSI Binding Operations

Applications use the NSI binding operations to export and import bindings to and from name service server entries. This group includes two suites of **_begin*( ), **_next*( ) and **_done*( ) routines, which applications can use to import bindings.

Function	Description
rpc_ns_entry_expand_name( )	Expands the name of a name service entry.
rpc_ns_entry_inq_resolution( )	Resolves the cell namespace components of a name and returns partial results.
rpc_ns_entry_object_inq_begin( )	Creates an inquiry context for viewing the objects stored in an entry in a name service database.
rpc_ns_entry_object_inq_done( )	Deletes a name service object inquiry context.
rpc_ns_entry_object_inq_next( )	Returns an object stored in an entry in a name service database.

Table 6-2 NSI Entry Operations

Applications use the NSI entry operations to return information about name service entries of various types.

Function	**Description**
rpc_ns_group_delete( )	Deletes a group attribute.
rpc_ns_group_mbr_add( )	Adds an entry name to a group; if necessary, creates the entry.
rpc_ns_group_mbr_inq_begin( )	Creates an inquiry context for viewing group members.
rpc_ns_group_mbr_inq_done( )	Deletes the inquiry context for a group.
rpc_ns_group_mbr_inq_next( )	Returns a member name from a group.
rpc_ns_group_mbr_remove( )	Removes an entry name from a group.

Table 6-3 NSI Group Operations

Applications use the NSI group operations to manipulate name service group entries.

Function	**Description**
rpc_ns_mgmt_binding_unexport( )	Removes multiple binding handles, or object UUIDs, from an entry in a name service database.
rpc_ns_mgmt_entry_create( )	Creates an entry in a name service database.
rpc_ns_mgmt_entry_delete( )	Deletes an entry from a name service database.
rpc_ns_mgmt_entry_inq_if_ids( )	Returns the list of interfaces exported to an entry in a name service database.
rpc_ns_mgmt_free_codesets( )	Frees a code sets array that has been allocated in memory.
rpc_ns_mgmt_handle_set_exp_age( )	Sets a handle's expiration age for cached copies of name service data.
rpc_ns_mgmt_inq_exp_age( )	Returns an application's global expiration age for cached copies of name service data.
rpc_ns_mgmt_read_codesets( )	Reads the code sets attribute associated with a server entry in the name service database.
rpc_ns_mgmt_remove_attribute( )	Removes an attribute from a server entry in the name server database.
rpc_ns_mgmt_set_attribute( )	Adds an attribute to a server entry in the name server database.
rpc_ns_mgmt_set_exp_age( )	Modifies the application's global expiration age for cached copies of name service data.

Table 6-4 NSI Management Operations

The NSI management operations carry out operations typically done by name service management applications or only infrequently done by most applications.

Function	**Description**
rpc_ns_profile_delete( )	Deletes a profile attribute.
rpc_ns_profile_elt_add( )	Adds an element to a profile; if necessary, creates the entry.
rpc_ns_profile_elt_inq_begin( )	Creates an inquiry context for viewing the elements in a profile.
rpc_ns_profile_elt_inq_done( )	Deletes the inquiry context for a profile.
rpc_ns_profile_elt_inq_next( )	Returns an element from a profile.
rpc_ns_profile_elt_remove( )	Removes an element from a profile.

Table 6-5 NSI Profile Operations

Applications use the NSI profile operations to manipulate name service profile entries.

CAE Specification

Part 3

Global Directory Service

The Open Group

Chapter 7

X.500 Services and Protocols

The Global Directory Service (GDS) is based on the Directory Service standard known as X.500 and specified in the joint ISO 9594 (1990) standard and CCITT X.500 series of recommendations. These documents specify both the services and the protocols used in X.500. Associated OSI services and protocol standards are also referenced. All references are listed below and detailed in **Referenced Documents** on page xvii.

Document	Date Published
ISO 8326	1987
ISO 8327	1987
ISO 8327-2 CD	1990
ISO 8649	1988
ISO 8650	1988
ISO 8650-2 DIS	1990
ISO 8822	1988
ISO 8823	1988
ISO 8823-2 DIS	1990
ISO 8824	1990
ISO 8825	1990
ISO / IEC 9072-1	1989
ISO / IEC 9072-2	1989
ISO / IEC 9594-1	1990
ISO / IEC 9594-2	1990
ISO / IEC 9594-3	1990
ISO / IEC 9594-4	1990
ISO / IEC 9594-5	1990
ISO / IEC 9594-6	1990
ISO / IEC 9594-7	1990
ISO / IEC 9594-8	1990
ISO / IEC 10021-2	1990
PUB 217	1992
PUB 218	1992
X.249	1992

Chapter 8

Conformance Statement for GDS

This chapter specifies the requirements for conforming GDS implementations. The specification is based on the Protocol Implementation Conformance Statement (PICS) Proformas for X.500 and the related protocol stack.

The following definitions apply to the contents of this chapter:

- An item is said to be *supported for origination* by an implementation if the implementation is able to generate it under some circumstances (either automatically or as a result of a related service being required by the user).
- An item is said to be *supported for reception* if it is correctly interpreted and handled, and, when required, made available to the user.

8.1 Notations and Abbreviations

In order to enhance the readability of the tables in this chapter, the following notations and abbreviations are used:

Status column

This column indicates the requirements of the Open Systems Testing Consortium PICS Proforma for DAP/DUA and DAP/DSA.

M (mandatory)	The operation or protocol element is a mandatory requirement for static conformance to the referenced capability.
O (optional)	The operation is an optional requirement for static conformance to the referenced capability.
- (not applicable)	There is no static requirement associated with the related item.

Support column

This column indicates whether or not the item is supported by GDS, or if the conformance statement is not applicable. It contains one of the following abbreviations:

Y (supported)	The item is supported by GDS.
N (not supported)	The item is not supported by GDS.
- (not applicable)	The conformance statement is either not applicable or not required.

Value column

The value or range of values in this column shall be implemented where relevant.

Note: The **Protocol Element** column in the Directory Access Protocol (DAP) tables (or, similarly, the first column in other tables) sometimes employs indentation to indicate subordination, as follows:

• (bullet)	indicates second level
— (dash)	indicates third level
simple indentation	indicates fourth and further levels.

Where subordination is indicated, the description of the superior element takes precedence over the subordinate element.

8.2 Directory Protocol Implementation Conformance

The Protocol Implementation Conformance Statement Proformas (PICS) for the Global Directory Service (GDS) consists of descriptions of the capabilities and options that are supported by the Directory User Agent (DUA) and Directory Service Agent (DSA) of the GDS. The PICS is designed for conformance test purposes. The PICS used for the X.500 protocols (ISO 9594-5) is based on the ones available from the Open Systems Testing Consortium (OSTC). Their scope is the DUA accessing the DSA in a stand-alone environment.

The PICS does not include statements about:

- strong and external credentials
- signing of values
- distributed directory features, namely:
  - — scoping of referrals
  - — continuation references.

There are no PICS available for the Directory System Protocol (DSP). The PICS used for upper OSI layers in X.500 (ACSE, ROSE, Presentation, Session) are based on ISO and CCITT work (see Chapter 7).

8.2.1 Support of Attribute Syntaxes, Attribute Types and Object Classes

Attribute syntaxes are defined in Table 8-1 for X.520, Table 8-2 for X.402, and Table 8-3 on page 63 for those that are private to GDS.

For explanations of the abbreviations used in the following tables, see Section 8.1 on page 60.

Attribute Syntax	Status	Support
Undefined Syntax	-	Y
Distinguished Name	O	Y
Object Identifier Syntax	O	Y
Case Exact String	O	Y
Case Ignore String	O	Y
Printable String	O	Y
Numeric String	O	Y
Case Ignore List	O	Y
Boolean Syntax	-	Y
Integer Syntax	-	Y
Octet String	O	Y
UTC Time	-	Y
Telephone Number Syntax	O	Y
Search Guide Syntax	-	Y
Postal Address Syntax	O	Y
Telex Number Syntax	O	Y
Teletex Terminal Identifier Syntax	O	Y
FAX Number Syntax	O	Y
Preferred Delivery Method Syntax	O	Y
Presentation Address Syntax	O	Y
Country Name Syntax	-	Y
Password Syntax	-	Y
Certificate Syntax	-	Y
Certificate Pair Syntax	-	Y
Certificate List Syntax	-	Y

Table 8-1 X.520 Attribute Syntaxes

Attribute Syntax	Support
MHS DL Submit Permission Syntax	Y
MHS O/R Address Syntax	Y
MHS O/R Name Syntax	Y
MHS Preferred Delivery Method Syntax	Y

Table 8-2 X.402 Attribute Syntaxes

Attribute Syntax	Support	Definition
Access Control List Syntax	Y	In addition to authentication, access protection is implemented for each object at attribute level.
ASN1	Y	This dummy-syntax may be used for all attributes whose syntax is not supported by the DSA. Attribute values are not decoded from ASN.1 representation to local format. No syntax checking is performed and no matching is allowed. This definition is the same as for ANY in ISO 8824.
Case Ignore IA5-String	Y	The valid character set is IA5-String for this syntax. Matching is as defined for Case Ignore String.

Table 8-3 Private Attribute Syntaxes

Attribute types are defined in Table 8-4 for X.520, Table 8-5 on page 65 for X.402, and Table 8-6 on page 65 for those that are private to GDS.

The following abbreviations are used in Table 8-4 to Table 8-6 inclusive:

Max Nbr. The maximum number of attribute values the implementation accepts. A **U** in this field means unlimited.

The values in this column are the values in the default GDS schema. This schema can be modified using GDS administration tools.

Max Size The maximum size (in octets) the implementation accepts. The maximum size can be expressed either as a maximum number of octets (here 180), or as a maximum number of lines (6 for this standard) of a given maximum of octets per line (here 30 for either T61 String or Printable String).

Table 8-4 X.520 Attribute Types

Attribute Type		Standard		Implemented		
		Max Nbr.	Max Size	Max Nbr.	Max Size	Support
System Attribute Types	Object-Class			U	28	Y
	Aliased-Object-Name	1		1	1024	Y
	Knowledge-Information			U	1024	Y
Labeling Attribute Types	Common-Name		64	2	64	Y
	Surname		64	2	64	Y
	Serial-Number		64	2	64	Y
Geographical Attribute Types	Country-Name	1	2†	1	2	Y
	Locality-Name		128	2	128	Y
	State-or-Province-Name		128	2	128	Y
	Street-Address		128	2	128	Y
Organizational Attribute Types	Organization-Name		64	2	64	Y
	Org.-Unit-Name		64	2	64	Y
	Title		64	2	64	Y

Attribute Type		**Standard**		**Implemented**		
		Max Nbr.	**Max Size**	**Max Nbr.**	**Max Size**	**Support**
Explanatory Attribute Types	Description		1024	U	1024	Y
	Search-Guide			U	256	Y
	Business-Category			2	128	Y
Postal Addressing Attribute Types	Postal-Address		6	2	180	Y
	T61-String		30	30	30	Y
	Printable-String		30	30	30	Y
	Postal-Code		40	2	40	Y
	Post-Office-Box		40	2	40	Y
	Phys.-Deliv.-Office-Name		128	2	128	Y
Telecommunications Addressing Attribute Types	Telephone-Number		32	U	32	Y
	Telex-Number			U	26	Y
	Telex-Number		14	14	14	Y
	Country-Code		4	4	4	Y
	Answerback		8	8	8	Y
	TTX-Terminal-Identifier			U		Y
	Teletex-Terminal		1024		1024	Y
	Parameters					Y
	Fax-Telephone-Number			U	37	Y
	Telephone-Number		32			Y
	Parameters					Y
	X121-Address		15	U	15	Y
	Internat.-ISDN-Number		16	U	16	Y
	Registered-Address		6	2	180	Y
	T61-String		30	30	30	Y
	Printable-String		30	30	30	Y
	Destination-Indicator		128	2	128	Y
Preferences Attribute Types	Preferred-Delivery-Method	1		1	40	Y
OSI Application Attribute Types	Presentation-Address			1	268	Y
	Suppl.-Applic.-Context			2	28	Y
Relational Attribute Types	Member			U	1024	Y
	Owner			U	1024	Y
	Role-Occupant			U	1024	Y
	See-Also			U	1024	Y
Security Attribute Types	User-Password		128	2	128	Y
	User-Certificate		3024	0	3024	Y
	CA-Certificate		3024	0	3024	Y
	Authority-Revocation-List		32503	0	32503	Y
	Certificate.-Revoc.-List		32503	0	32503	Y
	Cross-Certificate-Pair		6056	0	6056	Y

† The size of a Country-Name value must be precisely 2.

Attribute Type		**Standard**		**Implemented**		
		Max Nbr.	**Max Size**	**Max Nbr.**	**Max Size**	**Support**
MHS Attributes	MHS-Deliv.-Cont.Length	1	4	1	4	Y
	MHS-Deliv.-Cont.-Types			4	28	Y
	MHS-Deliverable-EITs			8	28	Y
	MHS-DL-Members			U	3596	Y
	MHS-DL-Submit-Permission			U	3604	Y
	MHS-Message-Store	1		1	1024	Y
	MHS-OR-Address			U	2564	Y
	MHS-Pref.-Deliv.-Meth.	1		1	40	Y
	MHS-Supp.-Autom.-Action			4	28	Y
	MHS-Supp.-Content-Types			4	28	Y
	MHS-Supp.-Optional-Attr.			U	28	Y

Table 8-5 X.402 Attribute Types

Attribute Type	**Syntax**	**Max Nbr.**	**Max Size**	**Support**
Master-Knowledge	Distinguished Name	1	1024	Y
Access-Control-List	Access Control List Syntax	1	20500	Y
Time-Stamp	UTC Time	1	18	Y
Structure-Rule-Table	Printable String	U	29	Y
Object-Class-Table	Printable String	U	397	Y
Attribute-Table	Printable String	U	101	Y
CDS-Cell	Octet String	1	284	Y
CDS-Replica	Octet String	1	905	Y
Principal-Name	Printable String	1	1024	Y
Authentication-Mechanism	Integer Syntax	4	4	Y
Alternate-Address	Octet String	1	800	Y

Table 8-6 Private Attribute Types

Object classes are listed in Table 8-7 for X.521, Table 8-8 for X.402, and Table 8-9 for object classes that are private to GDS.

Object Class	**Status**	**Support**
Top	-	Y
Alias (or subclass of alias)	O	Y
Country	O	Y
Locality	O	Y
Organization	O	Y
Organizational-Unit	O	Y
Person	-	Y
Organizational-Person	O	Y
Organizational-Role	-	Y
Group-of-Names	O	Y
Residential-Person	-	Y
Application-Process	-	Y
Application-Entity	-	Y
Directory-Service-Agent	-	Y
Device	-	Y
Strong-Auth.-User	-	Y
Certification-Authority	-	Y

Table 8-7 X.521 Object Classes

Object Class	**Support**
MHS-Distribution-List	Y
MHS-Message-Store	Y
MHS-Mess-Transfer-Agent	Y
MHS-User	Y
MHS-User-Agent	Y

Table 8-8 X.402 Object Classes

Name	**Subclass of**	**Must Contain**	**May Contain**
GDS-Top	Top		Master-Knowledge Access-Control-List
Schema	GDS-Top	Common-Name	Time-Stamp Structure-Rule-Table Object-Class-Table Attribute-Table

Table 8-9 Other Implemented Object Classes

8.2.2 DAP Protocol Implementation Conformance

Table 8-10 shows the global conformance statement for the Directory Access Protocol (DAP). Bind, unbind and directory operations are included.

The following tables are also included in the DAP conformance statement:

- Table 8-11
- Table 8-12 on page 68
- Table 8-24 on page 73
- Table 8-25 on page 73
- Table 8-26 on page 74.

For explanations of the abbreviations used in the following tables, see Section 8.1 on page 60.

Operation	Status	Support
DirectoryBind	M	Y
DirectoryUnbind	M	Y
Read	M	Y
Compare	M	Y
Abandon	M	Y
List	M	Y
Search	M	Y
AddEntry	M	Y
RemoveEntry	M	Y
ModifyEntry	M	Y
ModifyRDN	M	Y

Table 8-10 Global Statement of Conformance for DAP

Table 8-11 contains the conformance requirements for elements of the Directory Access Service beyond or outside the requirements for the DAP.

	Status	Support
Can the DSA contain Alias entries, and handle them appropriately?	O	Y
Can the DSA contain Replicated entries, and handle them appropriately?	O	Y
(ROSE) Does the DSA support operation Class 2 (asynchronous operations)?	M	Y
Does the DSA have the capability to act as a first level DSA?	-	Y

Table 8-11 Various Extra Requirements

	Status	**Support**
Does the DSA support Signing of:		
arguments	-	N
results	-	N

Table 8-12 Requirements for Signing

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
DirectoryBind Argument	M	Y	-	-	-	-	M	Y
• credentials	O	Y	-	-	-	-	O	Y
- simple	O	Y	-	-	-	-	O	Y
name	M	Y	-	-	-	-	M	Y
validity	-	N	-	-	-	-	-	N
time1	-	N	-	-	-	-	-	N
time2	-	N	-	-	-	-	-	N
random1	-	N	-	-	-	-	-	N
random2	-	N	-	-	-	-	-	N
password	O	Y	-	-	-	-	O	Y
- strong	-	N	-	-	-	-	-	N
- external procedure	-	N	-	-	-	-	-	N
• versions	M	Y	-	-	-	-	M	Y
DirectoryBind Results	-	-	M	Y	M	Y	-	-
• credentials	-	-	O	Y	O	Y	-	-
- simple	-	-	O	Y	O	Y	-	-
name	-	-	M	Y	M	Y	-	-
validity	-	-	-	N	-	N	-	-
time1	-	-	-	N	-	N	-	-
time2	-	-	-	N	-	N	-	-
random1	-	-	-	N	-	N	-	-
random2	-	-	-	N	-	N	-	-
password	-	-	O	Y	O	N	-	-
- strong	-	-	-	N	-	N	-	-
- external procedure	-	-	-	N	-	N	-	-
• versions	-	-	M	Y	M	Y	-	-
DirectoryBind Error	-	-	M	Y	M	Y	-	-
• versions	-	-	O	Y	O	Y	-	-
• Service-Error	-	-	M	Y	M	Y	-	-
- unavailable	-	-	M	Y	M	Y	-	-
• Security-Error	-	-	M	Y	M	Y	-	-
- inappropriate-authentication	-	-	M	Y	M	Y	-	-
- invalid-credentials	-	-	M	Y	M	Y	-	-

Table 8-13 Requirements on *DirectoryBind*

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
DirectoryUnbind Argument	-	-	-	-	-	-	-	-
DirectoryUnbind Result	-	-	-	-	-	-	-	-

Table 8-14 Requirements on *DirectoryUnbind*

No conformance requirements are specified for *DirectoryUnbind*.

Directory Operations

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
Read Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• Entry-Information-Selection	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
Read Result	-	-	M	Y	M	Y	-	-
• Entry-Information	-	-	M	Y	M	Y	-	-
- Object-Name	-	-	M	Y	M	Y	-	-
- From-Entry	-	-	O	Y	O	Y	-	-
- SET OF CHOICE	-	-	M	Y	M	Y	-	-
Attribute-Type	-	-	M	Y	M	Y	-	-
Attribute	-	-	M	Y	M	Y	-	-
• Common-Results	-	-	M	Y	M	Y	-	-

Table 8-15 Requirements on the *Read* Operation

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
Compare Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• Purported	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
Compare Result	-	-	M	Y	M	Y	-	-
• Object-Name	-	-	M	Y	M	Y	-	-
• Matched	-	-	M	Y	M	Y	-	-
• From-Entry	-	-	M	Y	M	Y	-	-
• Common-Results	-	-	M	Y	M	Y	-	-

Table 8-16 Requirements on the *Compare* Operation

Protocol Element	**DUA**				**DSA**				**Comment**
	Transmit		**Receive**		**Transmit**		**Receive**		
	Status	**Support**	**Status**	**Suppor.**	**Status**	**Support**	**Status**	**Support**	
Abandon Argument	M	N	-	-	-	-	M	Y	-
• Invoke-ID	M	N	-	-	-	-	M	Y	-
Abandon Result	-	-	M	N	M	Y	-	-	The DSA returns an Abandon-Failed error with Problem cannot-abandon.

Table 8-17 Requirements on the *Abandon* Operation

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
List Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
List Result	-	-	O	Y	O	Y	-	-
• List-Info	-	-	M	Y	M	Y	-	-
- Object-Name	-	-	M	Y	M	Y	-	-
- Subordinates	-	-	M	Y	M	Y	-	-
RDN	-	-	M	Y	M	Y	-	-
Alias-Entry	-	-	O	Y	O	Y	-	-
From-Entry	-	-	O	Y	O	Y	-	-
- Partial-Outcome-Qualifier	-	-	M	Y	M	Y	-	-
Limit-Problem	-	-	M	Y	M	Y	-	-
time-limit-exceeded	-	-	O	Y	O	Y	-	-
size-limit-exceeded	-	-	O	Y	O	Y	-	-
administrative-limit-exceeded	-	-	O	Y	O	Y	-	-
Unexplored	-	-	-	Y	-	Y	-	-
Unavailable-Crit-Ext	-	-	-	N	-	N	-	-
- Common-Results	-	-	M	Y	M	Y	-	-
• Uncorrelated-List-Info	-	-	O	N	O	N	-	-

Table 8-18 Requirements on the *List* Operation

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
Search Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• Subset	M	Y	-	-	-	-	M	Y
• Filter	M	Y	-	-	-	-	M	Y
- Filter-Item	M	Y	-	-	-	-	M	Y
equality	M	Y	-	-	-	-	M	Y
substrings	M	Y	-	-	-	-	M	Y
greater-or-equal	M	Y	-	-	-	-	M	Y
less-or-equal	M	Y	-	-	-	-	M	Y
present	M	Y	-	-	-	-	M	Y
approximate-match	M	Y	-	-	-	-	M	Y
- and	M	Y	-	-	-	-	M	Y
- or	M	Y	-	-	-	-	M	Y
- not	M	Y	-	-	-	-	M	Y
• Search-Aliases	M	Y	-	-	-	-	M	Y
• Selection	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
Search Result	-	-	M	Y	M	Y	-	-
• Search-Info	-	-	M	Y	M	Y	-	-
- Object-Name	-	-	M	Y	M	Y	-	-
- Entries	-	-	M	Y	M	Y	-	-
SET OF Entry-Information	-	-	M	Y	M	Y	-	-
Object-Name	-	-	O	Y	O	Y	-	-
From-Entry	-	-	O	Y	O	Y	-	-
SET OF CHOICE	-	-	M	Y	M	Y	-	-
Attribute-Type	-	-	M	Y	M	Y	-	-
Attribute	-	-	M	Y	M	Y	-	-
- Partial-Outcome-Qualifier	-	-	M	Y	M	Y	-	-
Limit-Problem	-	-	M	Y	M	Y	-	-
Unexplored	-	-	-	Y	-	Y	-	-
Unavailable-Crit-Ext	-	-	-	N	-	N	-	-
- Common-Results	-	-	M	Y	M	Y	-	-
• Uncorrelated-Search-Info	-	-	-	N	-	N	-	-

Table 8-19 Requirements on the *Search* Operation

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
AddEntry Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• Entry	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
AddEntry Result	M	Y	-	-	-	-	M	Y

Table 8-20 Requirements on the *AddEntry* Operation

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
RemoveEntry Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
RemoveEntry Result	M	Y	-	-	-	-	M	Y

Table 8-21 Requirements on the *RemoveEntry* Operation

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
ModifyEntry Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• Changes	M	Y	-	-	-	-	M	Y
- ADD_ATTRIBUTE	M	Y	-	-	-	-	M	Y
- REMOVE_ATTRIBUTE	M	Y	-	-	-	-	M	Y
- ADD_VALUES	M	Y	-	-	-	-	M	Y
- REMOVE_VALUES	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
ModifyEntry Result	M	Y	-	-	-	-	M	Y

Table 8-22 Requirements on the *ModifyEntry* Operation

Protocol Element	**DUA**				**DSA**			
	Transmit		**Receive**		**Transmit**		**Receive**	
	Status	**Support**	**Status**	**Support**	**Status**	**Support**	**Status**	**Support**
ModifyRDN Argument	M	Y	-	-	-	-	M	Y
• Object	M	Y	-	-	-	-	M	Y
• New-RDN	M	Y	-	-	-	-	M	Y
• Delete-Old-RDN	M	Y	-	-	-	-	M	Y
• Common Arguments	M	Y	-	-	-	-	M	Y
ModifyRDN Result	M	Y	-	-	-	-	M	Y

Table 8-23 Requirements on the *ModifyRDN* Operation

Common Elements

Protocol Element	DUA				DSA				Comment
	Transmit		Receive		Transmit		Receive		
	Status	Support	Status	Support	Status	Support	Status	Support	
Service Controls	O	Y	-	-	-	-	O	Y	-
• options	O	Y	-	-	-	-	O	Y	-
- Prefer-Chaining	-	Y	-	-	-	-	-	Y	-
- Chaining-Prohib	-	Y	-	-	-	-	-	Y	-
- Local-Scope	-	Y	-	-	-	-	-	Y	-
- Dont-Use-Copy	-	Y	-	-	-	-	-	Y	-
- Dont-Dereference-Aliases	M	Y	-	-	-	-	M	Y	-
• Priority	-	Y	-	-	-	-	-	Y	Accepted and ignored
• Time-Limit	O	Y	-	-	-	-	O	Y	-
• Size-Limit	O	Y	-	-	-	-	O	Y	-
• Scope-Of-Referral	-	Y	-	-	-	-	-	Y	-
Security Parameters	-	N	-	-	-	-	-	N	-
• Cert-Path	-	-	-	-	-	-	-	-	-
• Name	-	-	-	-	-	-	-	-	-
• Time	-	-	-	-	-	-	-	-	-
• Random	-	-	-	-	-	-	-	-	-
• Target	-	-	-	-	-	-	-	-	-
Requestor	O	N	-	-	-	-	O	Y	Accepted and ignored
Operation-Progress	-	Y	-	-	-	-	-	Y	-
Aliased-RDNs	-	Y	-	-	-	-	-	Y	-
Extensions	-	N	-	-	-	-	-	N	-

Table 8-24 Requirements on Elements in Common Arguments

Protocol Element	DUA				DSA			
	Transmit		Receive		Transmit		Receive	
	Status	Support	Status	Support	Status	Support	Status	Support
Security Parameters	-	-	-	N	-	N	-	-
• Cert-Path	-	-	-	-	-	-	-	-
• Name	-	-	-	-	-	-	-	-
• Time	-	-	-	-	-	-	-	-
• Random	-	-	-	-	-	-	-	-
• Target	-	-	-	-	-	-	-	-
Performer	-	-	O	N	O	N	-	-
Alias-Dereferenced	-	-	O	Y	O	Y	-	-

Table 8-25 Requirements on Elements in Common-Results

Protocol Element	DUA				DSA			
	Transmit		Receive		Transmit		Receive	
	Status	Support	Status	Support	Status	Support	Status	Support
Attribute-Error	-	-	M	Y	M	Y	-	-
• Object-Name	-	-	M	Y	M	Y	-	-
• Problems	-	-	M	Y	M	Y	-	-
- Problem	-	-	M	Y	M	Y	-	-
- Attribute-Type	-	-	M	Y	M	Y	-	-
- Attribute-Value	-	-	M	Y	M	Y	-	-
Name-Error	-	-	M	Y	M	Y	-	-
• Problem	-	-	M	Y	M	Y	-	-
• Matched	-	-	M	Y	M	Y	-	-
Referral	-	-	M	Y	M	Y	-	-
• Candidate	-	-	M	Y	M	Y	-	-
Abandoned	-	-	M	N	M	Y	-	-
Security-Error	-	-	M	Y	M	Y	-	-
• Problem	-	-	M	Y	M	Y	-	-
Service-Error	-	-	M	Y	M	Y	-	-
• Problem	-	-	M	Y	M	Y	-	-
- busy	-	-	M	Y	M	Y	-	-
- unavailable	-	-	M	Y	M	Y	-	-
- unwilling-to-perform	-	-	M	Y	M	Y	-	-
- chaining-required	-	-	-	N	-	N	-	-
- unable-to-proceed	-	-	-	Y	-	Y	-	-
- invalid-reference	-	-	-	Y	-	Y	-	-
- time-limit-exceeded	-	-	O	Y	O	Y	-	-
- administrative-limit-exceeded	-	-	-	Y	-	Y	-	-
- loop-detected	-	-	-	Y	-	Y	-	-
- unavailable-critical-extension	-	-	-	Y	-	Y	-	-
- out-of-scope	-	-	-	Y	-	Y	-	-
- dit-error	-	-	-	N	-	N	-	-
Abandon-Failed	-	-	M	N	M	Y	-	-
• Problem	-	-	M	-	M	Y	-	-
- no-such-operation	-	-	M	-	M	N	-	-
- too-late	-	-	M	-	M	N	-	-
- cannot-abandon	-	-	M	-	M	Y	-	-
• Operation	-	-	M	-	M	Y	-	-
Update-Error	-	-	M	Y	M	Y	-	-
• Problem	-	-	M	Y	M	Y	-	-
- naming-violation	-	-	O	Y	O	Y	-	-
- object-class-violation	-	-	O	Y	O	Y	-	-
- not-allowed-on-non-leaf	-	-	O	Y	O	Y	-	-
- entry-already-exists	-	-	O	Y	O	Y	-	-
- affects-multiple-DSAs	-	-	-	Y	-	Y	-	-
- object-class-modification-prohibited	-	-	O	Y	O	Y	-	-

Table 8-26 Requirements on Errors and Parameters

8.2.3 DSP Protocol Implementation Conformance

No PICS are available for the Directory System Protocol (DSP). GDS supports the DSP protocol as defined in ISO 9594-5.

8.2.4 ACSE Protocol Implementation Conformance

Table 8-27 to Table 8-29 on page 76 inclusive contain the PICS for the Association Control Service Element (ACSE).

For explanations of the abbreviations used in these tables, see Section 8.1 on page 60.

Protocol Mechanism	Status	Support
Normal-Mode	O	Y
X.410-1984-Mode	O	N
Rules-For-Extensibility	M	Y
Supports-Operation-Of-Session-V2	O	Y

Table 8-27 Requirements for Supported Functions

APDU	Transmit		Receive	
	Status	Support	Status	Support
A-associate-request APDU (AARQ)	M	Y	M	Y
A-associate-response APDU (AARE)	M	Y	M	Y
A-release-request APDU (RLRQ)	M	Y	M	Y
A-release-response APDU (RLRE)	M	Y	M	Y
A-abort APDU (ABRT)	M	Y	M	Y

Table 8-28 Requirements for Normal Mode APDUs

APDU Parameter	**Transmit**		**Receive**		**Value & Comment**
	Status	**Support**	**Status**	**Support**	
AARQ					
• Protocol-Version	M	Y	M	Y	-
• Application-Context-Name	M	Y	M	Y	-
• Calling-AP-Title	O	Y	O	Y	-
• Calling-AE-Qualifier	O	N	O	N	accepted
• Calling-AP-Invokation-Identifier	O	N	O	N	accepted
• Calling-AE-Invokation-Identifier	O	N	O	N	accepted
• Called-AP-Title	O	Y	O	Y	-
• Called-AE-Qualifier	O	N	O	N	accepted
• Called-AP-Invokation-Identifier	O	N	O	N	accepted
• Called-AE-Invokation-Identifier	O	N	O	N	accepted
• Implementation-Information	O	N	M	N	accepted
• User-Information	M	Y	M	Y	-
AARE					
• Protocol-Version	M	Y	M	Y	-
• Application-Context-Name	M	Y	M	Y	-
• Responding-AP-Title	O	Y	O	Y	-
• Responding-AE-Qualifier	O	N	O	N	accepted
• Responding-AP-Invokation-Identifier	O	N	O	N	accepted
• Responding-AE-Invokation-Identifier	O	N	O	N	accepted
• Result	M	Y	M	Y	-
• Result-Source-Diagnostic	M	Y	M	Y	-
• Implementation-Information	O	N	M	N	accepted
• User-Information	M	Y	M	Y	-
RLRQ					
• Reason	M	Y	M	Y	Transmit specify Reason = normal (0)
• User-Information	M	Y	M	Y	-
RLRE					
• Reason	M	Y	M	Y	Transmit specify Reason = normal (0)
• User-Information	M	Y	M	Y	-
ABRT					
• Abort-Source	M	Y	M	Y	-
• User-Information	O	Y	O	Y	-

Table 8-29 Requirements for Normal Mode Parameters

8.2.5 ROSE Protocol Implementation Conformance

Table 8-30 to Table 8-32 on page 78 inclusive contain the PICS for the Remote Operation Service Elements (ROSE).

For explanations of the abbreviations used in the following tables, see Section 8.1 on page 60.

Class	Code	Support
Association Class for DAP	1	Y
Association Class for DSP	3	Y
Operation Class for DAP	1 or 2	Y
Operation Class for DSP	2	Y

Table 8-30 Requirements for Association and Operation Class

APDU	Transmit		Receive	
	Status	Support	Status	Support
RO-Invoke APDU (ROIV)	M	Y	M	Y
RO-Result APDU (RORS)	M	Y	M	Y
RO-Error APDU (ROER)	M	Y	M	Y
RO-Reject APDU (RORJ)	M	Y	M	Y

Table 8-31 Requirements for Supported APDUs

APDU Parameter	Transmit		Receive		Value & Comment
	Status	Support	Status	Support	
ROIV					
• Invoke-ID	M	Y	M	Y	-
• Linked-ID	M	Y	M	Y	The Directory does not use the Linked-ID parameter.
• Operation-Value	M	Y	M	Y	-
• Argument	O	Y	O	Y	Transmit Argument is Directory Operation Argument.
RORS					
• Invoke-ID	M	Y	M	Y	-
• Operation-Value	O	Y	O	Y	-
• Result	O	Y	O	Y	Transmit Result is Directory Operation Result.
ROER					
• Invoke-ID	M	Y	M	Y	-
• Error-Value	M	Y	M	Y	-
• Error-Parameter	O	Y	O	Y	-
RORJ					
• Invoke-ID	M	Y	M	Y	-
- Invoke-ID-Type	O	Y	M	Y	-
- NULL	M	Y	M	Y	-
• Problem (choice of)	M	Y	M	Y	-
- General-Problem	M	Y	M	Y	All values (0-2) supported.
- Invoke-Problem	M	Y	M	Y	All values (0-7) supported.
- Return-Result-Problem	O	Y	M	Y	All values (0-2) supported.
- Return-Error-Problem	O	Y	M	Y	All values (0-4) supported.

Table 8-32 Requirements for Supported Parameters

8.2.6 Presentation Service Elements Protocol Implementation Conformance

Table 8-33 to Table 8-36 on page 80 inclusive contain the PICS for the Presentation Service Elements.

For explanations of the abbreviations used in the following tables, see Section 8.1 on page 60.

Mode	Status	Support
Normal	O	Y
X.410-1984	O	N

Table 8-33 Requirements for Supported Protocol Mechanisms

Functional Unit	Status	Support	Comment
Kernel	M	Y	-
Presentation-Context-Management	O	N	Not used by X.500.
Presentation-Context-Restoration	-	N	Not used by X.500.

Table 8-34 Requirements for Supported Functional Units

PPDU	Transmit		Receive		Comment
	Status	Support	Status	Support	
CP	M	Y	M	Y	-
CPA	M	Y	M	Y	-
CPR	M	Y	M	Y	-
ARU	M	Y	M	Y	-
ARP	M	Y	M	Y	-
TD	M	Y	M	Y	-
TE		-		-	Not used by X.500.
TTD		-		-	Not used by X.500.
TC		-		-	Not used by X.500.
TCC		-		-	Not used by X.500.

Table 8-35 Requirements for PPDUs (Kernel Function Unit)

PPDU Parameter	Transmit		Receive		Value & Comment
	Status	**Support**	**Status**	**Support**	
CP					
• Calling-Presentation-Selector	O	Y	M	Y	-
• Called-Presentation-Selector	O	Y	M	Y	-
• Mode-Selector	M	Y	M	Y	normal mode
• Presentation-Context-Definition-List	O	Y	M	Y	abstract-syntax-names, transfer-syntax-names (up to five names)
• Default-Context-Name	O	N	M	Y	-
• Protocol-Version	O	Y	M	Y	-
• Presentation-Requirements	O	N	M	Y	kernel, In Transmit all bits = 0
• User-Session-Requirements	O	N	M	Y	kernel, duplex
• User-Data	O	Y	M	Y	-
CPA					
• Responding-Presentation-Selector	O	Y	M	Y	-
• Mode-Selector	M	Y	M	Y	normal mode
• Presentation-Context-Definition-Result-List	O	Y	M	Y	-
• Protocol-Version	O	Y	M	Y	-
• Presentation-Requirements	O	Y	M	Y	kernel, In Transmit all bits = 0
• User-Session-Requirements	O	N	M	Y	kernel, duplex
• User-Data	O	Y	M	Y	-
CPR					
• Response-Presentation-Selector	O	Y	M	Y	-
• Presentation-Context-Definition-Result-List	O	Y	M	Y	-
• Default-Context-Result	O	Y	M	Y	provider rejection
• Protocol-Version	O	Y	M	Y	-
• Provider-Reason	O	Y	M	Y	-
• User-Data	O	Y	M	Y	-
ARU					
• Presentation-Context-Identifier-List	O	Y	M	Y	-
• User-Data	O	Y	M	Y	-
ARP					
• Provider-Reason	O	Y	M	Y	-
• Event-Identifier	O	Y	M	Y	-
TD					
• User-Data	O	Y	M	Y	-

Table 8-36 Requirements for PPDU-Parameters (Kernel Function Unit)

8.2.7 Session Service Elements Protocol Implementation Conformance

Table 8-37 to Table 8-40 on page 82 inclusive contain the PICS for the Session Service Elements.

For explanations of the abbreviations used in the following tables, see Section 8.1 on page 60.

Functional Unit	Status	Support	Comment
Kernel	M	Y	-
Negotiated-Release	O	N	Not used by X.500
Half-Duplex	O	N	Not used by X.500
Duplex	O	Y	-
Expedited-Data	O	N	Not used by X.500
Typed-Data	O	N	Not used by X.500
Capability-Data-Exchange	O	N	Not used by X.500
Minor-Synchronize	O	N	Not used by X.500
Symmetric-Synchronize	O	N	Not used by X.500
Major-Synchronize	O	N	Not used by X.500
Resynchronize	O	N	Not used by X.500
Exceptions	O	N	Not used by X.500
Activity-Management	O	N	Not used by X.500

Table 8-37 Requirements for Supported Functional Units

Protocol Mechanism	Status	Support
Use of transport expedited data (Extended Control Quality of Service)	O	N
Refuse of transport connection (sending)	O	N
Refuse of transport connection (receiving)	O	Y
Basic concatenation	M	Y
Extended concatenation (sending)	O	N
Extended concatenation (receiving)	O	N
Segmenting (sending)	O	N
Segmenting (receiving)	O	N
Segmenting for unlimited user data (sending)	O	N
Segmenting for unlimited user data (receiving)	O	N

Table 8-38 Requirements for Supported Protocol Mechanisms

SPDU	Transmit		Receive		Comment
	Status	**Support**	**Status**	**Support**	
Connect SPDU (CN)	M	Y	M	Y	-
Accept SPDU (AC)	M	Y	M	Y	-
Refuse SPDU (RF)	M	Y	M	Y	-
Finish SPDU (FN)	O	Y	M	Y	-
Disconnect SPDU (DN)	O	Y	O	Y	-
Abort SPDU (AB)	M	Y	M	Y	-
Data Transfer SPDU (DT)	O	Y	M	Y	-
Connect Data Overflow SPDU (CDO)	O	-	O	-	Not used by X.500.
Overflow Accept SPDU (CDO)	O	-	O	-	Not used by X.500.
Abort Accept SPDU (CDO)	O	-	O	-	Not used by X.500.
Prepare SPDU (CDO)	-	-	-	-	Not used by X.500.

Table 8-39 Requirements for SPDUs (Kernel Function Unit)

Table 8-40 Requirements for SPDU Parameters (Kernel Function Unit)

SPDU Parameter	Transmit		Receive		Value & Comment
	Status	**Support**	**Status**	**Support**	
Connect					
• Connection-Identifier	O	Y	M	Y	-
- PGI-Default (absent)	O	Y	M	Y	-
- PGI-Default (empty)	O	Y	M	Y	-
- Calling-SS-User-Reference	O	Y	M	Y	-
- Common-Reference	O	Y	M	Y	-
- Additional-Reference-Information	O	Y	M	Y	-
• Connect-Accept-Item	O	Y	M	Y	-
- PGI-Default (absent)	O	Y	M	Y	-
- PGI-Default (empty)	O	Y	M	Y	-
- PGI-Default (not empty)	O	Y	M	Y	-
- Protocol-Options	M	Y	M	Y	-
- TSDU-Maximum-Size	O	N	M	Y	-
- Version-Number	M	Y	M	Y	-
- Initial-Serial-Number	O	Y	M	Y	-
- Token-Setting-Item	O	Y	M	Y	-
- Second-Initial-Serial-Number	-	N	-	N	-
• Session-User-Requirements	O	Y	M	Y	-
• Calling-SSAP-Identifier	O	Y	M	Y	-
• Called-SSAP-Identifier	O	Y	M	Y	-
• User-Data	O	Y	M	Y	max size is 512 octets
• Data-Overflow	O	N	M	Y	-
• Extended-User-Data	O	N	M	Y	max size is 10240 octets

SPDU Parameter	**Transmit**		**Receive**		**Value & Comment**
	Status	**Support**	**Status**	**Support**	
Accept					
• Connection-Identifier	O	Y	M	Y	-
- PGI-Default (absent)	O	Y	M	Y	-
- PGI-Default (empty)	O	Y	M	Y	-
- Called-SS-User-Reference	O	Y	M	Y	-
- Common-Reference	O	Y	M	Y	-
- Additional-Reference-Information	O	Y	M	Y	-
• Connect-Accept-Item	O	Y	M	Y	-
- PGI-Default (absent)	O	Y	M	Y	-
- PGI-Default (empty)	O	Y	M	Y	-
- PGI-Default (not empty)	O	Y	M	Y	-
- Protocol-Options	M	Y	M	Y	-
- TSDU-Maximum-Size	O	N	M	Y	-
- Version-Number	M	Y	M	Y	-
- Initial-Serial-Number	O	Y	M	Y	-
- Token-Setting-Item	O	Y	M	Y	-
- Second-Initial-Serial-Number	-	N	-	N	-
• Token-Item	O	Y	M	Y	-
• Session-User-Requirements	O	Y	M	Y	-
• Calling-SSAP-Identifier	O	Y	O	Y	-
• Called-SSAP-Identifier	O	Y	M	Y	-
• User-Data	O	Y	M	Y	max size is 10240 octets
• Enclosure-Item	-	N	-	N	-
Refuse					
• Connection-Identifier	O	Y	M	Y	-
- PGI-Default (absent)	O	Y	M	Y	-
- PGI-Default (empty)	O	Y	M	Y	-
- Called-SS-User-Reference	O	Y	M	Y	-
- Common-Reference	O	Y	M	Y	-
- Additional-Reference-Information	O	Y	M	Y	-
• Transport-Disconnect	O	N	M	Y	-
• Session-User-Requirements	O	Y	M	Y	-
• Version-Number	O	Y	M	Y	-
• Reason-Code	M	Y	M	Y	-
• Enclosure-Item	-	N	-	N	-
Finish					
• Transport-Disconnect	O	N	M	Y	-
• User-Data	O	Y	M	Y	max size is 10240 octets
• Enclosure-Item	-	N	-	N	-
Disconnect					
• User-Data	O	Y	M	Y	max size is 10240 octets
• Enclosure-Item	-	N	-	N	-

SPDU Parameter	Transmit		Receive		Value & Comment
	Status	Support	Status	Support	
Abort					
• Transport-Disconnect	M	Y	M	Y	-
• Reflect-Parameter-Values	O	Y	M	Y	-
• User-Data	O	Y	M	Y	max size is 10240 octets
• Enclosure-Item	-	N	-	N	-
Data Transfer					
• Enclosure-Item	-	N	-	N	-
• User-Information-Field	O	Y	M	Y	-

CAE Specification

Part 4

Cell Directory Service

The Open Group

Chapter 9

CDS Service Definition

This chapter contains the abstract service definition of the Cell Directory Service (CDS).

The scope of CDS within the DCE Directory Service is best described by a cell. CDS consists of a set of facilities that control the directory service entries within the cell of a cell name space.

The common semantics and syntax rules for cell name spaces are defined in Chapter 1. Conformance to this CDS specification includes conformance to the Directory Service Information Model specified in Chapter 1.

Implementations that conform to this specification interoperate according to the following rule: any conforming CDS clerk implementation interoperates with any conforming CDS server or GDA implementations, and any conforming CDS server implementations interoperate.

9.1 Name Syntax

The syntax for names presented to CDS is specified in Section 1.3 on page 11. Before names are processed by CDS, they are converted to their canonical form as specified in **Canonicalisation of Names** on page 14.

The canonical string representation of names defines the wire format of both the atomic name component and the composite name. Its encodings are specified in the data structures **cds_Name_t** and **cds_FullName_t** in Section 11.2.2 on page 125 and Section 11.2.3 on page 125 respectively.

9.1.1 Filters for Enumerate Operations

The general matching rules for names are specified in Section 1.3.4 on page 17.

If a component of a name passed as input to a CDS enumeration operation contains one of the wildcard metacharacters (that is, unescaped * (asterisk) or ? (question mark)), and if this component is the terminal component of a CDS name, then the following filter algorithm is applied to the component:

- The parent directory of the wildcarded name component is searched for any matching entry name (either object, child pointer or soft link), by:
  - — substituting any occurrence of a ? (question mark) metacharacter with exactly one character of the entry name
  - — substituting any occurrence of a * (asterisk) metacharacter with a substring (zero or multiple characters) of the entry name.
- The complete set of entry names that pass the filter is said to be matched.

9.2 Functional Model

The functional model of CDS, as described in this chapter, includes the modular composition of CDS, its integration with other DCE services, a list of the facilities that are excluded from this specification, and a description of the security model of CDS.

Implementations intended to comply with the **X/Open DCE** shall adhere to this specified functional model. The internal design of facilities may vary, but they shall provide the defined semantics to ensure interoperability.

CDS is functionally divided into a number of major modules:

- CDS client
- CDS clerk
- CDS server, including the Transaction Agent and Clearinghouses.

Figure 9-1 illustrates the relationship between these modules and shows the appropriate communication protocols. The Global Directory Agent, although not directly part of CDS, is also accessed by the CDS clerk and shown in the figure.

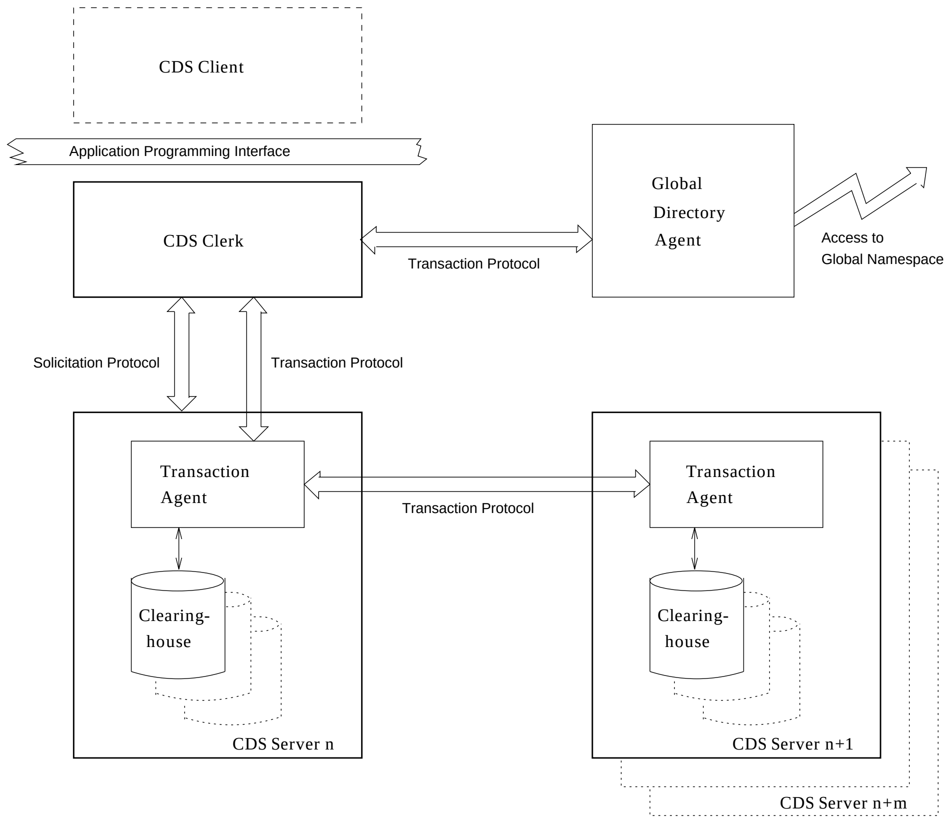

Figure 9-1 CDS Functional Modules

The following sections summarise the functions of these CDS modules. For further information, see Section 9.4 on page 106 for CDS services specifications, Chapter 10 for the protocol specification and Section 2.2 on page 21 for the Global Directory Agent specification.

Note: The Global Directory Agent provides for inter-cell operations by connecting the cell name spaces to the global name space and, in the case of cell hierarchies, to other cell name spaces within the hierarchy. It does not provide generic access to GDS. For information about accessing and manipulating the X.500 GDS and the X.500 conformance statements refer to Chapter 3, Chapter 4 and Part 3, Global Directory Service.

9.2.1 CDS Client and Clerk

A client accesses CDS through the client application programming interface. This interface is provided by a CDS module called *clerk*.

Note: The immediate API to CDS is not further specified in **X/Open DCE**. A higher-level programming interface, the X/Open Directory Service (XDS) interface, is specified in Chapter 3.

The clerk is the only CDS module that must reside on all DCE nodes. The clerk ascertains an appropriate CDS server to process a request, and then invokes the clerk/server operations of the *transaction protocol* to communicate to as many CDS servers (including the Global Directory Agent) as necessary to satisfy the request. The clerk presents the client's login context to the RPC interface.

Clerks are responsible for initially learning about at least one CDS server that is able either to process a complete request or provide information (referrals) about other candidate CDS servers. This initial information is obtained via the *solicitation protocol*. CDS servers periodically advertise their availability using the RPC broadcast execution semantic. Clerks listen for these CDS server advertisements and thereby learn of the existence of servers and name spaces.

To improve responsiveness when CDS is initialised, a clerk is also permitted (although not required) to issue a single solicitation request (using the RPC broadcast execution semantic) to provoke announcements from CDS servers that may be waiting to advertise.

In order to maximise performance, it is likely that a clerk implementation maintains a cache of recently accessed information; the algorithm for managing such a cache is not specified in this document. Also, the clerk may execute a number of management operations to initialise itself; these operations are not further specified in this document.

9.2.2 CDS Server

Each CDS consists of one or multiple instances of servers that provide the services requested by clerks. CDS servers communicate with clerks and other servers via the transaction protocol.

The *transaction agent* is the main CDS server module. It processes the CDS server side of the transaction protocol, performing operations requested by clients. The transaction agent is also responsible for coordinating the creation, deletion and modification of directories, using the server-to-server operations of the transaction protocol for communicating with other transaction agents, as necessary. In order to perform these maintenance operations, the server accesses other servers in a manner identical to the way clients access servers — namely, through a clerk. Thus, each CDS server logically contains a clerk which provides all the functions of the normal client clerk, plus a number of special functions relevant only to server-to-server communication.

The CDS server supports the solicitation protocol specified in Chapter 10. Servers send advertisements under the following circumstances:

- when the advertisement timer expires
- when a solicitation request is received
- when the server is enabled.

CDS stores name space data in a partitioned and possibly partially replicated database. The database is partitioned because parts of the name space are stored in different locations. The database may be partially replicated because parts of the name space may be simultaneously stored in multiple locations. The unit of both partitioning and replication is the directory; a collection of directories stored on a particular node is called a *clearinghouse*. Clearinghouses are integral to CDS servers. Partitioning is accomplished by controlling which directories are stored in which clearinghouse. Replication is accomplished by storing a directory in more than one clearinghouse.

attributes that are specified in All clearinghouses are catalogued in object entries in the cell root directory. Clearinghouses thus have names so that clients may conveniently refer to them, and so that CDS itself can find clearinghouses by looking up their names in the name space. Information which controls the operation of the clearinghouse as a whole is stored in a *pseudo-directory* entry in the clearinghouse. This clearinghouse pseudo-directory contains the same attributes as a normal directory, as well as a number of additional Section 9.3 on page 96.

Any copy of a directory (including the original copy) stored in a particular clearinghouse is called a *replica*. In order to simplify the algorithms for name creation and general name space maintenance, one of the replicas of a directory must be designated to be the *master replica*.

Conforming CDS implementations may provide replication services that maintain other types of replicas (such as read-only replicas). These services must ensure that the contents of all replicas of a directory remain consistent.

Further control functions of the CDS server, such as the overall coordination of the server operation (bringing clearinghouses on line, and so on) are implementation-specific and thus not specified in this document.

9.2.3 Global Directory Agent

The Global Directory Agent (GDA) handles CDS clerk requests for CDS entries that are not local to the requesting cell (that is, any global compound name that does not match the name of the local cell). The GDA returns an updated progress record whose replica pointers (on successful GDA look-up operations) contain information about the appropriate clearinghouses in the targeted foreign cell. This is true as well if the targeted foreign cell is, in fact, another cell within a cell hierarchy.

Any CDS clerk may treat the GDA as a special-case server which it invokes to resolve the global compound name of an unresolved fully-qualified name. The fact that a replica set returned by a CDS server identifies a set of GDAs rather than CDS servers is transparent to the clerk. In the case mentioned here, the global root /... may be returned in the progress record as the resolved part of the name, and the remaining composite name as the unresolved part of the name.

This behaviour is based on the assumption that the clerk has been referred to, or *a priori* knows of, a CDS server that holds a replica of the cell's root directory. The directory entry is referenced by the cell's root directory via a GDA pointer (instead of a child pointer entry) containing a set-valued attribute **CDS_GDAPointers**. Each value in **CDS_GDAPointers** identifies a GDA.[5]

Progress records returned by the GDA only return partial results; the **PR_done** flag in the progress record always has the value FALSE.

The status returned by the GDA can be one of the following:

[CDS_SUCCESS]
: If the unresolved name (residual) in the progress record is not empty, the client's operation request can proceed to resolve it.

  If the unresolved name in the progress record becomes empty, and the type is not **PR_directory**, it should be treated as [CDS_UNKNOWNENTRY], as all cell names must be directories.

[CDS_UNKNOWNENTRY]
: The requested entry could not be found (for example, DNS returned error status [NXDOMAIN]).

[CDS_UNDERSPECIFIEDNAME]
: The composite name of the request contained an existing entry in the global name space, but the specified object is not a cell name.

[CDS_NOTSUPPORTED]
: The name service (GDS, DNS, or CDS) is not reachable, or the requested operation is unsupported.

9.2.4 Requirements on Components Specified Elsewhere

This section specifies the external requirements of CDS for other services, as specified in other volumes of **X/Open DCE**. The usage of CDS by other services is specified in the service specifications.

Remote Procedure Call

CDS utilises RPC for communication, as specified in the **DCE Remote Procedure Call** specification. CDS uses authenticated RPC (protection level may be protect_level_none, for instance, for the clerk initialisation, at which an unauthenticated RPC requests the cell configuration information in the name space).

The protocols are encoded as RPC interfaces in IDL notation. Protocol Data Units (PDUs) are represented as operations of RPC interfaces.

5. The child pointer for the global root is generated by initialising the GDA by means of the *cds_CreateChild* operation.

Time

CDS relies on the distributed time provided by the DCE Distributed Time Service, as specified in the **DCE Time Services** specification, to maintain synchronised clocks in the network for use in the sequencing of updates in the name space. It is not guaranteed that the CDS protocol works correctly if server clocks differ by more than five minutes.

Security

CDS uses the Security Service and authenticated RPC, as specified in the **DCE Security Services** specification and the **DCE Remote Procedure Call** specification respectively, for mutual authentication and authorisation of CDS entities.

CDS clerks, the mediators between clients and servers, are authenticated by impersonating their clients' identity.

CDS servers of a cell are registered in an authorisation group which puts all cell servers within a single protection boundary. This allows for obtaining the required credentials in inter-server communication. Servers accept peer update requests from servers who are recognised as peers.

CDS entities are protected by Access Control Lists that are maintained and verified through the Common ACL Manager interface, as specified in the **DCE Security Services** specification.

9.2.5 Facilities and Features Excluded from the CDS Architecture

CDS accommodates the architecture as outlined in Chapter 1. Thus, the following features and facilities are explicitly excluded from the specification of CDS (implementations may include these features and facilities, but no conformance guarantees are made about them whatsoever):

Typed Names
CDS does not recognise any form of typed names directly, nor is the simulation of typed names using attribute-based searches of the cell name space efficient.

Local Names
CDS is not designed completely to replace the local names used by operating systems and applications to refer to their abstract objects.

Distributed Database
CDS is not designed as a generalised distributed database and must not be used as such. It provides few of the usual capabilities associated with database systems, and offers very weak consistency guarantees to its clients, in order to provide high levels of replication and rapid access.

CDS does not guarantee things such as external consistency, atomic transactions, and so on.

Storing in Cache
The model and algorithm for storing relevant information in the cache at the CDS clerks is not specified. However, it is likely that implementations make use of sophisticated storing in cache for improved efficiency.

9.2.6 Access Control and Protection Model

CDS uses the DCE Security Service for controlling user access to its protected objects. Protected objects consist of the following:

- all clerk and server instances
- all name entries that are stored in clearinghouses — that is, all directories, soft links, object entries and child pointers.

These objects are protected by Access Control Lists (ACLs), as specified in the **DCE Security Services** specification.

ACLs attached to CDS clerks and servers exist for maintenance purposes, and are local to those facilities.

ACL Manager Types Supported by CDS

The protected objects of CDS are supported by its ACL Managers, which are derived from the Common ACL Manager. For the algorithm used by the Common ACL Manager, and for further details on the access control model, refer to the **DCE Security Services** specification. The CDS ACL Manager types are as follows:

Objects

Controls ACLs on CDS object entries. This ACL manager type supports the following permission set:

read
write
delete
test
control.

The UUID of this ACL manager type is:

```
c2e7e53c-4455-11ca-99bd-08002b1c8f1f
```

Directories

Controls ACLs on CDS directory entries. This ACL manager type supports the following permission set:

read
write
delete
test
control
insert
admin.

The UUID of this ACL manager type is:

```
d1a74194-4455-11ca-b064-08002b1c8f1f
```

Clearinghouses

Controls ACLs on CDS clearinghouse entries. This ACL manager type supports the following permission set:

read
write
delete
test
control.

The UUID of this ACL manager type is:

```
d645d095-4455-11ca-b028-08002b1c8f1f
```

Server management

Controls ACLs on CDS servers. This ACL manager type supports the following permission set:

read
write
control.

The UUID of this ACL manager type is:

```
faf2e540-58b8-11ca-a04a-08002b12a70d
```

Clerk management

Controls ACLs on CDS clerks. This ACL manager type supports the following permission set:

read
write
control.

The UUID of this ACL manager type is:

```
dc8c6fc0-6143-11ca-b4b9-08002b1bb4f5
```

ACL Types

To protect both terminal object and directory entries, and to enable newly created entries automatically to inherit default ACLs from their parent directory, the CDS ACL facility supports the following DCE ACL types. For further information on these ACL types as defined by the access control model, refer to the **DCE Security Services** specification.

Object ACL

The Object ACL type grants permissions for any CDS name (object entries, soft links, child pointers, clearinghouses and directories), as well as to clerks and servers. When associated with a CDS directory, the permissions granted with the Object ACL type apply only to the directory itself, not to the directory's contents or to any child directories.

Initial Object Creation ACL

The Initial Object Creation ACL type applies only to CDS directory names. This ACL type defines the initial permissions specifically for a directory's future contents (soft links, application-defined object entries, child pointers and clearinghouse object entries). The permissions granted using the Initial Object Creation ACL type apply only to the future contents of the directory, not to the directory itself. The permissions are propagated only to names that are created in the directory after the ACL entry is created; permissions are not propagated to names that already exist in the directory.

Initial Container Creation ACL

The Initial Container Creation ACL type applies only to CDS directory names. This ACL type defines the initial permissions for a directory that automatically propagate (by default) to all child directories that may later be created under that directory. The permissions granted using the Initial Container Creation ACL type apply only to the child directories that are created after the ACL entry is created; permissions are not propagated to child directories that already exist.

ACL Entry Types

ACL entry types are used to specify the category of principal for which the ACL entry is created. See the **DCE Security Services** specification for a description of DCE ACL entry types.

Permissions Supported by CDS

CDS supports the following DCE permissions:

printstring	**bit representation**
Read	0x00000001
Write	0x00000002
Control	0x00000008
Insert	0x00000010
Delete	0x00000020
Test	0x00000040
Administer	0x00008000

Each permission has a slightly different meaning, depending on the kind of CDS name with which it is associated. In general, the permissions are defined as follows:

Read Allows a principal to look up a name and view the attribute values associated with it.

Write Allows a principal to change the modifiable attributes associated with a name, except for its ACLs.

Control Allows a principal to modify the ACL entries associated with a name. (Control permission is automatically granted to the creator of a CDS name.)

Insert Allows a principal to create new names in a directory. (For use within directory entries only.)

Delete Allows a principal to delete a name from the name space controlled by CDS.

Test Allows a principal to test whether an attribute of a name has a particular value, without being able actually to see any of the values (that is, without having read permission for the name).

Administer Allows a principal to issue CDS commands that control the replication of directories. (For use with directory entries only.)

The required permissions for successful invocation of RPC interface operations on CDS protected objects are specified in Chapter 12 of this document.

CDS Servers Authorisation Group and Cell Creation

Inter-server communication runs within one protection boundary. Each CDS server of a particular cell is registered in a single authorisation group, and is permitted access to peer servers through its group permission rights.

This authorisation group has the well known relative name **subsys/dce/cds-server** at the top level in the security name space. Whenever a CDS server is created, it must be added to this group.

The cell creation procedure must grant full permission on the cell root directory to this group. Object ACL and Initial Container Creation ACL entries are also created specifying **subsys/dce/cds-server** as the principal in each ACL entry, to ensure that the group has full access to all future directories and their contents.

9.3 Architected Default Attributes

All CDS name entries (objects, directories, soft links and child pointers) consist of a set of *attributes* and their associated value or values. For more information on attributes and their names, types and identifiers, refer to Chapter 1.

Table 9-1 on page 97 lists the operational attributes defined by CDS, together with the following information about them: whether the attribute is single-valued or set-valued; whether it is read-only or modifiable by a client (a CDS server may modify any attribute); its usage for different entry types; and whether the attribute is always or optionally present. The object identifiers associated with these attributes are listed in Appendix B. The encodings of the attribute syntaxes are defined in Chapter 11. The sections following the table describe each of the attributes in detail.

Table 9-1 on page 97 enforces a schema for operational attributes used by CDS. Any attribute that has only read permission and is associated with a specified set of entry types cannot be created and used on any entry type. Attributes that have write permission and are only specified for specific entry types, however, can be used by applications in entry types that do not use these as operational attributes. This specification does not define the meaning of these attribute names, if used by applications.

In Table 9-1 on page 97, the following abbreviations are used in the **Permissions** column:

R	read
W	write
D	delete.

Attribute			Per-	Usage	
Name	**Type**	**Syntax**	**missions**	**Entry Type**	**Presence**
CDS_CTS	Single	VT_Timestamp	R	All	Mandatory
CDS_UTS	Single	VT_Timestamp	R	All	Mandatory
CDS_Class	Single	VT_byte	RW	Object	Optional
CDS_ClassVersion	Single	VT_Version	RW	Object	Optional
CDS_ObjectUUID	Single	VT_uuid	RW *	Object	Optional
			R	Directory, Child Pointer	Mandatory
CDS_Replicas	Set	VT_ReplicaPointer	R	Directory, Child Pointer	Mandatory
CDS_AllUpTo	Single	VT_Timestamp	R	Directory	Mandatory
CDS_Convergence	Single	VT_small	RW	Directory	Mandatory
CDS_InCHName †	Single	VT_small	R	Directory	Conditional
CDS_ParentPointer	Set	VT_ParentPointer	R	Directory	Mandatory
CDS_DirectoryVersion	Single	VT_Version	R	Directory, Clearinghouse	Mandatory
CDS_UpgradeTo †	Single	VT_Version	RWD	Directory	Conditional
CDS_LinkTarget	Single	VT_FullName	RW	Soft Link	Mandatory
CDS_LinkTimeout	Single	VT_Timeout	RW	Soft Link	Mandatory
CDS_Towers	Set	VT_byte	RWD	Object, Clearinghouse	Optional
CDS_CHName	Set	VT_FullName	R	Clearinghouse	Mandatory
CDS_CHLastAddress	Single	VT_byte	R	Clearinghouse	Mandatory
CDS_CHState	Single	VT_small	R	Clearinghouse	Mandatory
CDS_CHDirectories	Set	VT_CHDirectory	R	Clearinghouse	Mandatory
CDS_ReplicaState	Single	VT_small	R	Directory	Mandatory
CDS_ReplicaType	Single	VT_small	R	Directory	Mandatory
CDS_LastSkulk	Single	VT_Timestamp	R	Directory	Mandatory
CDS_LastUpdate	Single	VT_Timestamp	R	Directory	Mandatory
CDS_RingPointer ††	Set	VT_uuid	R	Directory	Conditional
CDS_Epoch	Single	VT_uuid	R	Directory	Mandatory
CDS_ReplicaVersion	Single	VT_Version	R	Directory	Mandatory
CDS_NSCellname	Single	VT_char	R	Clearinghouse	Mandatory
CDS_GDAPointers †	Set	VT_gdaPointer	R	Directory	Conditional
CDS_CellAliases	Set	VT_GroupMember	RWD	Directory	Conditional
CDS_ParentCellPointers	Single	VT_ReplicaPointer	RWD	Directory	Conditional
RPC_ClassVersion	Single	VT_byte	RWD	Object	Optional
RPC_ObjectUUIDs	Single	VT_byte	RWD	Object	Optional
RPC_Group	Set	VT_byte	RWD	Object	Optional
RPC_Profile	Set	VT_byte	RWD	Object	Optional
RPC_Codesets	Set	VT_byte	RWD	Object	Optional
SEC_RepUUID	Single	VT_byte	R	Object	Conditional

Table 9-1 Operational Attribute Summary

* The **CDS_ObjectUUID** attribute is writable only once, during creation time.

† The **CDS_InCHName**, **CDS_UpgradeTo** and **CDS_GDAPointers** attributes are present only in cell root directory entries.

†† The **CDS_RingPointer** attribute is obsolete in DCE 1.1.

9.3.1 CDS_CTS Attribute

CDS_CTS (Creation Time Stamp) is a single-valued attribute which is present and non-null in every entry. It contains a space- and time-unique handle on the entry, which is assigned when the entry is made and which is never changed. This attribute is also used as a timestamp marking the creation time of the entry.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.2 CDS_UTS Attribute

CDS_UTS (Update Time Stamp) is a single-valued attribute which is present and non-null in any entry that has been updated. The **time** portion of the UTS gives the time at which the most recent update to any attribute of the entry took place. For object entries, the UTS gives the timestamp of the most recent update to any attribute of the object entry. For directory entries, the UTS gives the timestamp of the most recent update to an attribute of the directory.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.3 CDS_Class Attribute

CDS_Class is a single-valued operational attribute which may be present and non-null in object entries. This attribute is used to classify object entries according to the type of object being named. Object classes have names that are printable strings (with a maximum length of 31 characters).

The usage of this attribute is application determined. The contents of this attribute are not specified in this document.

Note that only object entries have a **CDS_Class** attribute. The only built-in class is for clearinghouse objects, identified as **CDS_Clearinghouse**.

9.3.4 CDS_ClassVersion Attribute

CDS_ClassVersion is a single-valued attribute which may be present and non-null in object entries. It allows the definition of an object class to be evolved over time (for example, by changing the definition of the class-specific attributes), without confusing the clients of the CDS. It consists of a one octet **major version**, and a one octet **minor version**. The setting and interpretation of the class version is the responsibility of the application that defined the corresponding object class.

9.3.5 CDS_ObjectUUID Attribute

CDS_ObjectUUID is a single-valued attribute which is present and non-null in any directory and child pointer entry; it may be present in an object entry.

Directory and child pointer entries use this attribute internally to store the UUID that uniquely identifies the entry.

On object entries it can optionally be used to store the UUID of the actual object being referenced by the CDS entry. It is the responsibility of clients to properly maintain this attribute for the object they are interested in; CDS makes no effort to ensure the correctness of the values stored in the **CDS_ObjectUUID** attribute for object entries.

On object entries, this attribute can be set only on creation, and is thereafter not modifiable.

9.3.6 CDS_Replicas Attribute

CDS_Replicas is a set-valued attribute which is present and non-null in every directory and child pointer entry. Each member of the set identifies one of the clearinghouses that stores a replica of this directory. The **CDS_Replicas** attribute may not be modified directly; the set is updated as a side effect of the operations which create, destroy or change the replication of a directory.

Each element of the set consists of four parts:

- a type value describing the capabilities of this copy of the directory-master or read-only replica
- the UUID of the clearinghouse which stores the replica
- the fully-qualified global name of the clearinghouse which stores the replica
- the protocol towers, giving a hint for locating and accessing the clearinghouse.

See the data type definition of **cds_ReplicaPointer_t** in Chapter 11 for the precise encoding of this attribute's values.

9.3.7 CDS_AllUpTo Attribute

CDS_AllUpTo is a single-valued attribute which gives a bound on how out of date various replicas of this directory are. This attribute is present in every directory entry. All replicas of a directory are guaranteed to have received all updates of entries whose timestamps are less (that is, older) than the value of **CDS_AllUpTo**.

This attribute is read-only across the clerk-to-server RPC interface.

The usage and maintenance of this attribute depends on the replication service.

9.3.8 CDS_Convergence Attribute

CDS_Convergence is a single-valued attribute which specifies how persistent a directory should be in trying to keep its replica up to date.

The usage and maintenance of this attribute depends on the replication service.

9.3.9 CDS_InCHName Attribute

CDS_InCHName is a single-valued attribute which indicates whether a directory or any of its descendants can store clearinghouse names. If this value is TRUE, the directory can store clearinghouse names. If it is FALSE, the directory cannot store clearinghouse names. CDS will create this attribute on the cell root directory and give it a value of TRUE, it will not appear in any other directory.

9.3.10 CDS_ParentPointer Attribute

Note: The directory and clearinghouse structures in **X/Open DCE** are specified for directory version 3.0. In addition, version 4.0 is required if either hierarchical cells or cell aliasing functionality is to be used. The protocol version for this document, however, is 1.0; it is expressed in the IDL definition and is not directly connected with the directory version identified by the **CDS_DirectoryVersion** attribute.

CDS_ParentPointer is a set-valued attribute which is present in every directory entry except the cell root (see below). It contains a set of pointers to each directory's parent in the name space tree. This attribute is maintained automatically by CDS servers in order to keep the graph of the name space properly connected at all times. Only one value for the set is supported currently.

The attribute links a child directory to its parent, allowing clerks and servers to work up the tree as well as down. The attribute is used to make sure that parent and child directories always point to each other during normal operation, and it is used during directory creation to allow a child directory to link itself into the tree by creating a child pointer entry in the parent directory.

Note: The cell root directory does not have a parent pointer. Its **CDS_GDAPointers** point to the GDA of that cell name space.

For directory version 3.0 (value of the **CDS_DirectoryVersion** attribute), this set-valued attribute contains a single value with the following information:

- the UUID of the parent directory
- the fully-qualified global name of the parent directory
- a timeout value after which the pointer must be checked to ensure that the child still points to the parent — this is used to update replica pointers in the child pointer.

CDS_ParentPointer attributes are periodically checked to make sure that the name space controlled by CDS remains connected and that failures are reported in a timely fashion.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.11 CDS_DirectoryVersion Attribute

CDS_DirectoryVersion records the current version of a directory. Multiple directory versions are supported in a name space in order to enable the graceful migration to newer versions. This document specifies directory version 3.0.

Note: Any DCE configuration that uses cell aliasing or hierarchical cell functionality must be version 4.0.

When a directory is created using a clearinghouse on a CDS server with version 3.0 (or higher) of the directory, the value of **CDS_DirectoryVersion** is determined by the value of this attribute in the pseudo-directory entry of the clearinghouse. Thus, a directory created in a clearinghouse operating in version 3.0 is initially version 3.0.

This attribute has an additional role when present on the clearinghouse pseudo-directory. There it controls the storage format and semantics of new directories created using that clearinghouse master as a replica.

9.3.12 CDS_UpgradeTo Attribute

CDS_UpgradeTo is a single-valued attribute which may or may not be present in the cell root directory entry. It is used to control the upgrading of a directory from one version of CDS to another. By modifying this attribute, the process of upgrading a directory to a newer version of CDS may be initiated. If the operation has not reached its commit point, the upgrade may also be terminated by simply deleting this attribute.

Implementations must ensure that upgrade operations (proceeding in the background) in each replica are completed before the value of **CDS_DirectoryVersion** reflects the upgraded status of the directory indicated by the **CDS_UpgradeTo** attribute.

9.3.13 CDS_LinkTarget Attribute

CDS_LinkTarget is a single-valued attribute which is present and non-null in every soft link entry. It contains the fully-qualified global name of the entry to which the soft link points.

Note: The target entry need not necessarily exist.

9.3.14 CDS_LinkTimeout Attribute

CDS_LinkTimeout is a single-valued attribute which is present and non-null in every soft link entry. It contains a timeout data structure with two fields, which contain the expiration and extension time of the soft link. The algorithm that applies to the interpretation of these timeout value fields is shown in Table 9-2. If the soft link, when checked, is found to point nowhere (is dangling), the link is deleted.

Expiration Field	Extension Field	Action at Expiration Timeout
zero	zero	The link never expires.
zero	non-zero	Invalid. The behaviour is undefined, but implementations may simply ignore the value.
non-zero	zero	Entry is deleted.
non-zero	non-zero	If target still exists, extends life of entry by value of extension field. Otherwise entry is deleted.

Table 9-2 Timeout Values Evaluation

9.3.15 CDS_Towers Attribute

CDS_Towers is a set-valued attribute which does not have to be present, or which may be null in any given object or clearinghouse entry. This attribute is further described in the **DCE Remote Procedure Call** specification.

9.3.16 CDS_CHName Attribute

CDS_CHName is a set-valued attribute which is present and non-null in every clearinghouse pseudo-directory entry. In directory version 3.0 (the value of the **CDS_DirectoryVersion** attribute is 3.0), this attribute may have only one value, namely the primary fully-qualified global name of the clearinghouse. A set of values is allowed in order to support operations such as renaming or merging cells.

The attribute is used for the following purposes:

- to check that CDS operations are in fact directed at the correct clearinghouse
- to register correctly the clearinghouse object when the clearinghouse is created or when it moves to another CDS server.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.17 CDS_CHLastAddress Attribute

CDS_CHLastAddress is a single-valued attribute which is present and non-null in every clearinghouse pseudo-directory entry. It stores the location information at which the clearinghouse most recently reported itself to the rest of the name space. For directory version 3.0 and higher (the value of the **CDS_DirectoryVersion** attribute is 3.0), this value is structured as a protocol tower.

The value of this attribute is used to determine whether the clearinghouse has moved to a new CDS server.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.18 CDS_CHState Attribute

CDS_CHState is a single-valued attribute which is present in every clearinghouse pseudo-directory entry.

Its value describes the current state of the clearinghouse, and can be one of:

- **new**
- **on**
- **dying**.

The value of this attribute is used to determine whether or not the CDS server should advertise this clearinghouse via the solicitation or advertising protocol, and whether it is permitted to respond to look-up requests on directories it stores.

9.3.19 CDS_CHDirectories Attribute

CDS_CHDirectories is a set-valued attribute which is present in every clearinghouse pseudo-directory entry. It contains one member for each directory replica stored in the clearinghouse. The attribute can also exist on clearinghouse objects which can be used when restoring damaged name spaces.

The contents of each member of the set consist of the UUID of a directory replica and its fully-qualified global name, which can be used to look up entries in the directory.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.20 CDS_ReplicaState Attribute

CDS_ReplicaState is a single-valued attribute which determines whether a directory replica can be accessed. Its value describes the current state of the clearinghouse, and can be one of:

- **new** (during replica creation)
- **on** (normal operation)
- **dying** (during replica deletion).

This attribute is read-only across the clerk-to-server RPC interface.

9.3.21 CDS_ReplicaType Attribute

CDS_ReplicaType is a single-valued attribute which indicates whether a directory is a master or read-only replica.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.22 CDS_LastSkulk Attribute

CDS_LastSkulk is a single-valued attribute which records the timestamp of the last skulk performed on this directory.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.23 CDS_LastUpdate Attribute

CDS_LastUpdate is a single-valued attribute which records the timestamp of the most recent change to any attribute of the directory replica, or any change to an entry within the replica.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.24 CDS_RingPointer Attribute

CDS_RingPointer is a set-valued attribute which specifies the UUID of a clearinghouse containing another replica of this directory.

This attribute is read-only across the clerk-to-server RPC interface.

Note: This attribute will appear on older directories but not on DCE 1.1 directories.

9.3.25 CDS_Epoch Attribute

CDS_Epoch is a single-valued attribute which identifies a particular instance of the directory.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.26 CDS_ReplicaVersion Attribute

CDS_ReplicaVersion is a single-valued attribute which identifies the version of a replica of the directory.

This attribute is read-only across the clerk-to-server RPC interface.

9.3.27 CDS_NSCellname Attribute

CDS_NSCellname is a single-valued attribute which is present in every clearinghouse pseudo-directory entry. It contains the canonical string representation of the cell name for the clearinghouse.

9.3.28 CDS_GDAPointers Attribute

CDS_GDAPointers is a set-valued attribute which is present and non-null in the cell root directory entry only. This attribute contains location information about the registered Global Directory Agents for that cell, in the same way as the **CDS_Replicas** attribute. The type field in replica pointers is always set to **RT_gda**.

This attribute also includes a timeout value which functions similarly to the timeout in **CDS_ParentPointer**.

9.3.29 CDS_CellAliases Attribute

CDS_CellAliases is a set-valued attribute which contains alias names for the cell. This is resident in the root directory when an alias name exists.

This attribute can be created, modified and deleted by resident applications.

Note: Any DCE configuration that uses cell aliasing functionality must be version 4.0 (the value of the **CDS_DirectoryVersion** attribute is 4.0).

9.3.30 CDS_ParentCellPointers Attribute

CDS_ParentCellPointers is a single-valued attribute which contains GDA style pointers to the parent cell. This attribute is resident in the root directory when the cell is a child in a hierarchical cell combination.

Note: Any DCE configuration that uses hierarchical cell functionality must be version 4.0 (the value of the **CDS_DirectoryVersion** attribute is 4.0).

9.3.31 RPC_ClassVersion Attribute

RPC_ClassVersion is a single-valued attribute which contains the current Name Service Interface (NSI) version. Ths attribute is created by the NSI but can be created, written, modified and deleted by resident applications.

9.3.32 RPC_ObjectUUIDs Attribute

RPC_ObjectUUIDs is a set-valued attribute which contains optional UUIDs of the referenced server objects. This attribute is created by the Name Service Interface but can be created, written, modified and deleted by resident applications.

9.3.33 RPC_Group Attribute

RPC_Group is a set-valued attribute which contains server object names or service group names for this service group. This attribute is created by the Name Service Interface but can be created, written, modified and deleted by resident applications.

9.3.34 RPC_Profile Attribute

RPC_Profile is a set-valued attribute which contains server providers compromising the configuration profiles. This attribute is created by the Name Service Interface but can be created, written, modified and deleted by resident applications.

9.3.35 RPC_Codesets Attribute

RPC_Codesets is a set-valued attribute which contains character code set information. This attribute is created by the Name Service Interface but can be created, written, modified and deleted by resident applications.

9.3.36 SEC_RepUUID Attribute

SEC_RepUUID is a set-valued attribute which contains the replica instance UUID. This is stored in the servers name space entry.

This attribute is read-only across the clerk-to-server RPC interface.

9.4 Abstract Definitions of CDS Service Primitives

CDS provides cell-level directory service primitives that:

- operate on attributes
- operate on object entries
- operate on directory entries
- operate on soft links
- locate available servers.

The sections that follow present abstract definitions of these service primitives. The concrete specifications of these primitives, and how they map to CDS directory operations that use the RPC interfaces, are specified in Chapter 11 and Chapter 12.

Every service primitive takes as an input parameter the fully-qualified global name of the directory service entry. The resolution of this name may provoke a series of steps (which may involve multiple remote operations) which is repeated until the target clearinghouse has been located. Partial results may be evaluated and lead to a completion of the operation.

Unless the underlying operations detect a failure (valid status codes are listed in Appendix C), the status returned is [CDS_SUCCESS].

9.4.1 Service Primitives for Manipulating Attributes

The service primitives for manipulating attributes perform operations on attributes of CDS entries.

These primitives can only operate on entries that already exist in the name space. The operations that manipulate the name space are specified in the appropriate subsections for object, directory and soft link entries.

Some attributes that are mandatory for CDS entries are precluded from modification by these service primitives; these attributes are maintained by CDS servers internally. Refer to Section 9.3 on page 96 for details.

A valid CDS entry is specified to be one of the following:

- any directory
- any clearinghouse
- any object
- any directory or object
- any child pointer
- any soft link.

Some service primitives support only a subset of the preceding list.

The following sections contain abstract descriptions of the CDS service primitives for manipulating attributes.

Enumerate Attribute

An *Enumerate Attribute* operation is initiated by a client. It returns a set containing the attribute identifiers of the specified entry.

This service primitive may incur multiple remote operations, depending on the length of the clerk's internal buffer.

The attribute identifiers are enumerated in the order of object identifiers.

Modify Attribute

A *Modify Attribute* operation is initiated by a client. It applies one update to one attribute of a specified entry. Attributes can be added or removed from an entry with this primitive.

Any successful modification to an entry also causes the **CDS_UTS** timestamp of that entry to be updated.

Note that certain directory attributes that are only maintained internally (by the CDS server) must not be modified by this operation; CDS servers should reject any attempts otherwise. Attempts to modify clearinghouse or soft link entries are illegal; none of their attributes are modifiable.

This service primitive can also initiate operations to remove an object entry from the name space, if the type of the attribute to be removed is defined as **AT_none**. Attempts to remove other entry types are illegal.

Read Attribute

A *Read Attribute* operation is initiated by a client, and returns a set containing the values of the specified attribute. The values are encapsulated in discriminated unions, containing the syntax specifier and value pair.

The attribute values are enumerated in timestamp order of member creation. Those with the oldest timestamps are returned first.

This service primitive may incur multiple remote operations for set-valued attributes, depending on the length of the clerk's internal buffer.

Test Attribute

A *Test Attribute* operation is initiated by a client, and returns a boolean value indicating whether the supplied value was one of the values of the specified attribute.

9.4.2 Service Primitives for Manipulating Object Entries

Service primitives for manipulating object entries invoke operations that create, delete or look up object entries in the name space.

Creation of an object entry implies the creation and initialization both of the mandatory attributes maintained by CDS servers, and of the optional operational attributes. The modification of these attributes (if permitted), and the creation, look-up and modification of application-defined attributes, is controlled by the service primitives that manipulate attributes.

The following sections contain abstract descriptions of the CDS service primitives for manipulating object entries.

Create Object

A *Create Object* operation is initiated by the client, and creates the requested object entry.

The entry's **CDS_CTS** and **CDS_UTS** attributes are generated automatically by the CDS server. The entry's optional attributes **CDS_Class**, **CDS_ClassVersion** and **CDS_ObjectUUID** can be supplied by the caller. If **CDS_Class** is supplied, **CDS_ClassVersion** must also be supplied with the appropriate value.

This service primitive succeeds only if the name of the object is new in the name space (that is, if no entry with this name of any type exists in the name space). Furthermore, an object can be created only if its immediate parent directory already exists.

Delete Object

A *Delete Object* operation is initiated by a client, and removes the specified object entry from the name space.

Enumerate Object

An *Enumerate Object* operation is initiated by a client, and returns a set containing the names and classes of objects in the specified directory. Only object names that match a supplied filter, consisting of an atomic name (possibly wildcarded) and (optionally) the object class, are returned. A **NULL** class matches a * (asterisk metacharacter) filter.

The objects are enumerated in lexical order, according to the ordering rules of the Portable Character Set (PCS). The ordering for extended character sets is not specified.

This service primitive may incur multiple remote operations, depending on the length of the clerk's internal buffer.

9.4.3 Service Primitives for Manipulating Directory Entries

Service primitives for manipulating directory entries invoke operations that create, delete or look up directory entries in the name space. Included are services that modify child pointer entries.

Creation of a directory entry implies the creation and initialization of mandatory attributes maintained by CDS servers. The modification of these attributes (if permitted), and the creation, look-up and modification of application-defined attributes, is controlled by the services that manipulate attributes.

The following sections contain abstract descriptions of the CDS service primitives for manipulating directory entries.

Create Directory

A *Create Directory* operation is initiated by a client, and creates the requested directory entry in a specified or default clearinghouse.

This service primitive succeeds only if the name of the directory is new in the name space (that is, if no entry with this name of any type exists in the name space). Furthermore, a directory can be created only if its immediate parent directory already exists. This service primitive specifies the master clearinghouse.

Successful completion of the operation ensures the eventual consistency of the name space; in particular, the child pointers in the parent directory are appropriately created.

Delete Directory

A *Delete Directory* operation is initiated by a client, and removes an existing directory from the name space.

Successful completion of the operation ensures the eventual consistency of the name space; in particular, the child pointers in the parent directory are appropriately updated.

Enumerate Children

An *Enumerate Children* operation is initiated by a client, and returns a set containing the names of child pointers in the specified directory. Only child pointers that match a supplied filter consisting of an atomic name (possibly wildcarded) are returned.

The child pointers are enumerated in lexical order, according to the ordering rules of the Portable Character Set (PCS). The ordering for extended character sets is not specified.

This service primitive may incur multiple remote operations, depending on the length of the clerk's internal buffer.

Create Child

A *Create Child* operation is initiated by a server on a client's behalf, and creates a child pointer entry in the specified directory. It returns the value of the **CDS_CTS** attribute of the new child pointer entry.

This service primitive is performed if a client attempts to create a new directory (using *Create Directory*).

Delete Child

A *Delete Child* operation is initiated by a server on a client's behalf, and deletes the specified child pointer entry in the specified directory.

This service primitive is performed if a client attempts to delete a directory (using *Delete Directory*).

9.4.4 Service Primitives for Manipulating Soft Links

Service primitives for manipulating soft link entries invoke operations that create, delete or look up soft link entries in the name space.

Creation of a soft link entry implies the creation and initialization of mandatory attributes maintained by CDS servers. The modification of these attributes (if permitted), and the creation, look-up and modification of application-defined attributes, is controlled by the services that manipulate attributes.

The following sections contain abstract descriptions of the CDS service primitives for manipulating soft links.

Create Soft Link

A *Create Soft Link* operation is initiated by a client, and creates the specified soft link entry.

This service primitive can optionally specify the lifespan of the newly created soft link as being one of the following three:

temporary Soft links controlled by a timeout value for **temporary** use are deleted by CDS servers after the elapse of its expiration time, and CDS clerks have to invalidate possibly cached information about the target of the soft link.

extended The life of temporary soft links can be **extended** indefinitely by supplying an extension factor along with the expiration time. After the elapse of the expiration time, the CDS server checks for the existence of the target. If the target still exists, the server adds the extension factor to the expiration time to obtain the new expiration time and extend the life of the soft link. The server repeats this procedure continuously until the target entry is deleted.

permanent If no expiration time is specified during soft link creation, the soft link's existence is **permanent** until it is explicitly deleted.

The target of a soft link may not be in existence at the creation time of the soft link. The existence of the target is not verified by CDS servers.

Delete Soft Link

A *Delete Soft Link* operation is initiated by a client, and removes the specified soft link entry from the name space.

Enumerate Soft Links

An *Enumerate Soft Links* operation is initiated by a client, and returns a set containing the names of soft links in the specified directory.

Only soft links that match a supplied filter consisting of an atomic name (possibly wildcarded) are returned.

The soft links are enumerated in lexical order, according to the ordering rules of the Portable Character Set (PCS). The ordering for extended character sets is not specified.

This service primitive may incur multiple remote operations, depending on the length of the clerk's internal buffer.

Resolve Name

A *Resolve Name* operation is initiated by a client, and returns the resolved fully-qualified global name of the target entry pointed to. It resolves the name by following a sequence of soft links as specified in the fully-qualified global name input parameter.

9.4.5 Service Primitives for Advertisement and Solicitation

The service primitives for advertisement and solicitation perform operations that are necessary in order to provide CDS clerks with binding information to available clearinghouses in the cell.

The following sections contain abstract descriptions of the CDS service primitives for advertisement and solicitation.

Advertise

An *Advertise* operation is initiated by a CDS server, and advertises the availability of the server's clearinghouses to a broadcast address, which is listened to by CDS clerks.

This service primitive is triggered either by a CDS server internal timer or a received solicitation request.

Solicit

A *Solicit* operation is initiated by a clerk, and broadcasts a request to CDS servers which triggers their advertisements. No values are returned.

Solicit Server

A *Solicit Server* operation is initiated by a clerk, and returns the cell name, cell root directory UUID, and clearinghouses of the specified server.

Chapter 10

CDS Protocol Definition

This chapter contains the abstract definition of the CDS transaction protocol and solicitation protocol whose encodings are specified in Chapter 11; it is a specification of the protocol machines employed by CDS clerks and CDS servers.

Conforming implementations shall comply with the behaviour defined in this chapter.

10.1 Clerk Operation

The CDS clerk communicates with one or more CDS servers (including the Global Directory Agent) for the processing of client requests.

A clerk performs the following functions:

- discovers the existence of servers for the local cell name space
- formulates requests to be sent to CDS servers for processing, based on calls by the client
- communicates with one or more CDS servers using the transaction protocol
- receives responses from CDS servers to previously issued requests, and returns the results to the clients who issued the calls
- maintains any credentials needed by a client for authentication purposes by CDS servers
- walks a tree of directories that comprise the relevant part of the cell when necessary for processing of a client request
- optionally maintains a cache of recently accessed information.

10.1.1 Solicitation and Clearinghouse Selection

When requested by a client to perform some CDS operation, the clerk must first select a clearinghouse that is likely to be able to process the request. It then has to bind to the appropriate CDS server controlling that clearinghouse.

Since the cell's name space database is partitioned and distributed among a number of clearinghouses, the clerk must first decide which clearinghouses are good candidates for being able to process a client request, and then select one of them. The algorithm for choosing a clearinghouse is internal to the implementation of a clerk and how it caches previously received information; only the solicitation protocol for learning about available clearinghouses is specified in this document.

The following assumptions can be made about the location of clearinghouses:

- The clerk only needs to contact one CDS server to be able to find any clearinghouse in the name space.
- All clearinghouses contain a replica of the cell root; therefore any clearinghouse is able to provide location hints for any other clearinghouses that are always registered in the root directory.

Regardless of the details of its particular implementation, a clerk has some or all of the following information at its disposal to use in choosing a clearinghouse:

- a cached replica set for the parent directory containing the entry of interest

  In this case, the clerk already knows the entire potential set of clearinghouses that might process the request.

- a cached replica set for an ancestor directory
- binding information for clearinghouses that have advertised via the solicitation protocol

  This and the next case allow the clerk to select any available clearinghouse that can provide further guidance. This initiates a process that follows referral pointers through the name space finally to locate the requested entity.

- statically configured binding information for CDS servers.

The algorithm for selecting among multiple possible clearinghouses and CDS servers is not specified. Implementations may follow some rules of probability that reduce the average transactions necessary, balance the load among servers, and account for network topology.

The process of binding and establishing communication with the CDS server that supports a particular clearinghouse is subject to the underlying RPC protocol. This process includes mutual authentication. The clerk generally has several ways of obtaining the appropriate protocol tower for the server. These include:

- finding it in previously cached information

  This information can only be regarded as a hint; the information may be out of date. The address can only be learned with a higher degree of certainty by looking up the clearinghouse object with a confidence level of **medium**.

- finding it in data returned in the updated **ReplicaPointer**
- getting it by looking up the clearinghouse object in the name space.

These methods are usually tried in the order listed above, in the interest of securing the best performance.

10.1.2 Referral Handling (Progress Record)

When a clearinghouse is requested (in the person of a CDS server) to perform a CDS operation via the transaction protocol, it attempts to get the clerk as close to the desired result as possible, either by performing the operation itself, or by directing the clerk to other clearinghouses (that is, CDS servers) that can provide further guidance. A **progress record** returned to the clerk indicates the status of the requested operation and contains whatever information the server was able to resolve. The clerk is responsible for repeating the request (this time to the new clearinghouse that it was referred to) for as many times as is necessary to complete the information provided in the progress record.

The progress record contains the following flags and other fields:

PR_done	This flag is initially set to FALSE by the clerk and subsequently set to TRUE by the server if the operation was completed successfully. Otherwise, if the returned flag is FALSE, the fully-qualified global name of the request could only partially be resolved; the resolved and residual (unresolved) parts of the fully-qualified global name are returned in the appropriate fields of the progress record.
PR_up	This flag is initially set by the clerk to TRUE, and is never reset by the clerk. As soon as a clearinghouse for an ancestor directory is found, the server sets this flag FALSE to prevent oscillation in the search later on. The server leaves this flag TRUE if the name points to another cell. The GDA then clears this flag if the foreign cell name is successfully looked up.
PR_linked	This flag is initially set to FALSE by the clerk and then set to TRUE whenever a soft link is followed by the clerk or server in resolving the name.
PR_hitLink	This flag is initially set to FALSE by the clerk and then set to TRUE by a server when it detects a soft link and returns control to the clerk to check for loops in the soft link graph. The clerk resets this flag to FALSE before each new call to a server.

PR_ignoreState This flag is set to TRUE by the clerk if it wishes the server to ignore the state of a directory/replica and return information regardless.

PR_directory This flag is set to TRUE by the server whenever the partial or full resolved name is found to be a directory; otherwise (including the case where the full resolved name is NULL) it is set to FALSE.

This flag is required for cache handling when the entry type **ET_dirOrObj** is requested, because otherwise the clerk does not know which entity (directory or object) was matched if a match is reported.

Timeout This field is used by servers to compute the minimum timeout for all soft links followed in resolving a name. This information can then be used by the clerk to determine its policy for storing in cache, and may also be returned to the client as an indication of how long the supplied name is valid.

Unresolved Name This field contains the portion of the original fully-qualified global name that has not yet been located by the servers. The servers are responsible for maintaining this field, and clerks use the value (if any) returned in it as input for subsequent requests for the given operation. The clerk initially sets this field to the requested fully-qualified global name argument of the operation. Generally the field contains one of the following values:

- the entire fully-qualified global name, when the clerk makes its first attempt to get the operation performed
- the entire fully-qualified global name, while the clerk is working up the tree towards the root of the cell
- the portion of the fully-qualified global name that represents the residual of the name (that is, the part that has yet to be resolved)
- the UUID of the last directory resolved
- an entirely new name, after the clerk has traversed a soft link.

Resolved Name This field contains the portion of the current name which has been successfully processed (resolved) by the servers. The name in this field may be different from the original name supplied by the client if a soft link was traversed. The clerk is not required to do anything with the information in this field, but it may use it for either of the following purposes:

- returning partial information to the client if the operation status is [CDS_UNKNOWNENTRY]
- storing additional information about intermediate names in cache when walking the tree or traversing soft links.

Replica Pointer This field contains a set of clearinghouses which are believed to have a good probability either of being able to perform the desired operation or of having information about other clearinghouses closer to ones that can actually perform the operation. The clerk initially sets this field to NULL.

10.1.3 Tree Walk Algorithm

Every client call to the clerk results in the clerk's interactively walking the tree of interest and causing the operation to be performed at an appropriate clearinghouse for the directory.

The abstract semantic of the tree walk algorithm is as follows:

1. Initialise the progress record and the soft link loop detector.
2. Select the initial clearinghouse.
3. Send request to clearinghouse.

   If the selected clearinghouse is unavailable and the request fails, the clerk selects another clearinghouse that is at its disposal (iterate through previous step). See also Section 10.1.1 on page 114.
4. Loop while **PR_done** flag of result is FALSE:
   - If a soft link is encountered (that is, the flag **PR_hitLink** is TRUE), check for loops.
     - If a loop is detected, return to the client with failure status [CDS_POSSIBLECYCLE].
     - If no loop is detected, reinitialise and restart clearinghouse list for resolving new name.
   - Choose a clearinghouse from the list that was returned in **Replica Pointer**, until the list is exhausted. If there are no more clearinghouses to select from, return to the client with failure status [CDS_NOCOMMUNICATION].
   - Re-request the operation, with updated progress record, from the selected clearinghouse.
5. If **PR_done** flag is TRUE, return the operation result to the client.

10.1.4 Loop Detection

The clerk is responsible for detecting loops in soft links while walking the name space tree. Any time a soft link is detected by the server, the transaction agent returns a progress record to the clerk with the **PR_hitLink** flag set to TRUE. When this occurs, the clerk has to determine whether a loop in the graph of soft links has been detected (see Section 10.1.3); if so, the clerk returns the failure status [CDS_POSSIBLECYCLE] to the calling client.

The algorithm for loop detection and determining the depth of cycles is implementation-specific.

10.2 Server Operation

The CDS server contains two basic functional modules: the *clearinghouse* and the *transaction agent*.

10.2.1 Clearinghouse

Clearinghouses are databases in which the contents of the cell name space controlled by a CDS are stored. Clearinghouses accommodate partitioning and (possibly) replication.

The naming of clearinghouses follows a set of rigid rules which ensure that name lookup cannot fail because the clearinghouse in which the directory is storing the object entry cannot be found. In particular, CDS ensures that the following clearinghouse invariant is never violated during normal operation:

- Each clearinghouse contains a replica of the cell root directory.

Clearinghouses are either up or down. When a clearinghouse is up at a given node, that node is acting as a CDS server for that clearinghouse, and may advertise its availability. A server may be controlling more than one clearinghouse simultaneously.

A clearinghouse contains the following types of information:

- operational information:
  — clearinghouse global information
- directories, which contain:
  — directory global information
  — objects, soft links and child pointers.

Information that controls the operation of the clearinghouse as a whole is stored in a pseudo-directory entry in the clearinghouse. The unique identifier (UUID) of this pseudo-directory has the same value as the UUID of the clearinghouse object itself. Thus, clearinghouses can be named and accessed the same way as regular directory entries by clients. The specific attributes applied to these clearinghouse pseudo-directories are defined in Section 9.3 on page 96.

Every clearinghouse is catalogued in an object entry within the cell root directory. When a server is walking the tree looking for a clearinghouse, it may encounter a clearinghouse object whose information it needs to use for building a referral to the actual clearinghouse. Clearinghouses are accessed by the transaction agents of CDS servers. A set of operational attributes (as determined in Section 9.3 on page 96) are exclusively maintained by CDS servers. Any other entries and attributes can be accessed by clerks by means of requests to CDS servers.

Clients can access the clearinghouse operational information by specifying the clearinghouse name with the entry type **ET_clearinghouse**.

10.2.2 Transaction Agent

The transaction agent is responsible for processing clerk transactions, reading and writing clearinghouses, and communicating with transaction agents at other servers.

The transaction agent processes the following two sets of the transaction protocol:

- the basic clerk/server operations for communicating with clerks
- the directory maintenance operations for synchronising the state of parent and child directories for creation and deletion.

Logically, there is a single transaction agent for each clearinghouse, which processes transactions destined for it. However, the transaction agent has to multiplex the transactions of many clerks simultaneously, and it must thus be able to manage multiple simultaneous execution contexts. Whether this is accomplished by multi-threaded code, multiple processes or other mechanisms is implementation-dependent.

CDS servers export the transaction agent interface with the object UUID of clearinghouses. Clerks bind to clearinghouses using these object UUIDs.

The common operations needed for accessing clearinghouses and processing transactions are described below, in **Common Operations of the Transaction Agent**. This is followed by a specification of the protocol state model for operations on directory entries. **Common Operations of the Transaction Agent**, in conjunction with Section 9.4 on page 106, fully defines the abstract algorithm applied to any other operations.

Common Operations of the Transaction Agent

Each transaction initiated by a clerk consists of a single request. The transaction agent is responsible for processing the request and generating an appropriate response. However, before any request can be processed, the transaction agent must perform the following functions:

1. Determine whether the clearinghouse requested by the clerk is available.

   If the clearinghouse is available, the transaction agent proceeds as described in the following steps. Otherwise, one of the following failure statuses is returned to the clerk:

   [CDS_CLEARINGHOUSEDOWN]
      If the clearinghouse is known currently to reside at this transaction agent's server.

   [CDS_UNKNOWNCLEARINGHOUSE]
      If the clearinghouse is not known to this transaction agent's server.

2. Find the requested entry in the clearinghouse; if this cannot be done, return the partial result, together with any possible location hints, in an updated progress record to the clerk.

   Since a clearinghouse may contain any part of a path through the name space to the directory which contains the entry of interest, the transaction agent implements a local version of the tree walk algorithm used by clerks. In particular, the transaction agent must perform the following activities:

   i. Determine whether the clearinghouse contains some portion of the path provided by the clerk.

      If it does not, the clearinghouse must either return success with no additional information, or, if the **PR_up** flag is set to TRUE by the clerk, return a **CDS_GDAPointer**. If the **CDS_GDAPointer** does not exist, a [CDS_ROOTLOST] status is returned.

   ii. The transaction agent then walks the tree down from the point specified by the clerk in the progress record supplied, until one of the following happens:

      — The entry requested by the clerk is reached.

      — A child pointer entry pointing to a different set of clearinghouses is encountered.

      — A soft link is encountered.

      — An access failure occurs, or a portion of the path is determined not to exist. In this case, the appropriate failure status is returned.

3. Ensure that the client has sufficient permission to access the entry.

   The transaction agent calls the ACL Manager (see Section 9.2.6 on page 93) to perform the access control verification. If access is denied, the transaction agent returns the appropriate failure status to the clerk and rejects this operation request.

4. After successful verification of access permission on the encountered entry, the transaction agent proceeds as follows, depending on the result in step ii above:

   — If the requested entry is encountered, a handle is generated that can be used to perform the requested operation on the entry, as specified below.

   — If a child pointer entry pointing to a different set of clearinghouses is encountered, the progress record is updated and returned to the clerk.

   — If a soft link is encountered, the target of the soft link is obtained, the progress record is re-initialised with the new name, and control is returned to the clerk after setting the **PR_hitLink** and **PR_linked** flags to TRUE.

After obtaining the handle for the target entry of the requested operation, the transaction agent processes the request according to its service definition as follows:

- If the processing is successful, the appropriate information is updated in the clearinghouse and the progress record, status and possibly other output information are updated and returned to the requesting clerk.

  The transaction agent is responsible for creating and updating the appropriate operational attributes associated with entries.

- Unless the requested operation induced a manipulation of directory entries, the transaction affects only a single clearinghouse. Otherwise, the transaction agent may invoke other remote operations on a different clearinghouse before the request can be completed successfully. This specific behaviour is defined below in **Transactions for Manipulating Directory Entries**.

- If a failure in processing the request is detected, the appropriate failure status is returned to the requesting clerk.

Transactions for Manipulating Directory Entries

The operations which modify directory entries may require the coordination of multiple clearinghouses to maintain consistency in the name space structure. In general, the clerk issues the client's request to one clearinghouse (that is, CDS server), and the transaction agent for that clearinghouse calls its clerk to perform special internal server transactions to manipulate state at the other clearinghouse.

The clerk operations that may cause operations of the transaction protocol, which are originated by the initially targeted server and the activities of the transaction agent, are as follows:

cds_CreateDirectory

This operation causes a new directory to be created. It is issued by the clerk to the transaction agent controlling the clearinghouse that is to contain the master replica of the new directory. This may not be the same clearinghouse which holds the master replica for the new directory's parent; hence synchronisation is needed.

The algorithm for creating a new directory is as follows (note that this is an asynchronous series of actions):

1. A new directory is created at the local clearinghouse, and its **CDS_ParentPointer** attribute is set to point to its (purported) parent directory. Note that it may be true

that some directory on the specified path to the new directory may not exist, or that the new directory has a name which conflicts with an already existing child directory. These problems are detected when the attempt is made to create the child pointer entry at the parent's master replica (see the next step).

2. The transaction agent attempts to create the child pointer entry in the parent directory by performing the *cds_CreateChild* operation. If this fails, the new directory entry is destroyed and the appropriate failure status is returned to the clerk.
3. If the child pointer entry is successfully created, the **CDS_ParentPointer** attribute is set to the actual name of the parent directory (resolving a possibly linked name).
4. The Access Control List for the directory is appropriately initialised.

cds_DeleteDirectory

This operation removes an existing directory. The directory must be completely empty (that is, no object, soft link or child pointer entries can exist in the directory), or the deletion fails, and the failure status [CDS_NOTEMPTY] is returned to the calling clerk. The operation is issued by the clerk to the transaction agent controlling the master replica of the directory.

The algorithm for deleting a directory proceeds as follows (note that this is an asynchronous series of actions):

1. The transaction agent attempts to delete the child pointer entry in the parent directory by performing the *cds_DeleteChild* operation. If this fails, the appropriate failure status is returned to the calling clerk.

   Note: This operation is performed asynchronously in the background. The directory is marked as dying, and a success status is returned to the clerk. Some time may have elapsed before the child pointer entry is actually deleted and the directory is marked dead.

2. If the child pointer entry is successfully deleted, the local replica of the directory is placed in the dead state for processing the clean-up.

Chapter 11

CDS Protocol Encodings

This chapter specifies the encodings for both the CDS transaction protocol and the solicitation protocol. The messages transmitted between clerks and servers, and inter-server messages are specified as remote operations defined in RPC IDL (Interface Definition Language).

The transaction and solicitation protocols are defined in the RPC IDL files **cds_clerkserver.idl** and **cds_solicit.idl** respectively. The CDS protocol encodings are defined in the RPC IDL file **cds_types.idl** and the include file **dns_records.h**. The IDL definitions can be found in Appendix D. The operations that are specified in Chapter 12 map to the definitions in this IDL specification. The ordering of the operation declarations in the IDL specification is relevant for determining the associated procedure numbers. For more information on the IDL notation refer to the **DCE Remote Procedure Call** specification.

The **cds_types.idl** file defines CDS-related data structures for use as argument types in the CDS RPC interfaces, so marshalling and unmarshalling issues take precedence and this is the only place they are used.

The **dns_record.h** file defines CDS-related data structures as they are represented on the local machine, so space and alignment issues take precedence. These are the usual data structures you will see throughout the CDS code.

This chapter specifies the relevant data types and constants for these IDL interface definitions. Unless noted otherwise, messages are encoded in DCE Transfer Syntax (NDR).

11.1 Architected Limits

There are two syntaxes for naming service names: the external ASCII name and the internal *opaque name*. The external name is the syntax used for the human-readable form of a name. The internal name is the syntax of the name as passed across the client interface with the naming service. The external name is designed for readability while the internal name is designed to be convenient to encode in programs, protocols, and databases.

The maximum lengths of internal, or opaque, names are specified as follows:

```
#define SIMPLENAMEMAX   255  /* Single component name max size */
#define FULLNAMEMAX     402  /* Internal fully-qualified  */
                             /* global name max size as represented
                             /* in an array of single component names */
#define ATTRIBNAMEMAX   31   /* Attribute name max size */
                             /* Same size limitation applies to */
                             /* class names */
```

The maximum length of external names is 1023 characters including the null terminator.

11.2 Encoding of Names

11.2.1 Opaque Names

Opaque names can be either *simple* or *full*. A simple opaque name is a single relative distinguished name (RDN). A full opaque name represents a complete list of RDNs relative to the global root.

The **SimpleName_t** structure defines an internal, opaque encoding of names of various kinds (entry, attribute, class) to be passed through the CDS protocol. This structure is defined in **dns_record.h** as follows:

```
typedef unsigned char byte_t;
typedef unsigned char byte_u[sizeof(byte_t)];
typedef unsigned char bytes_u;

typedef struct {
        byte_u          sn_flag;
        byte_u          sn_length;
        bytes_u         sn_name[SIMPLENAMEMAX];
} SimpleName_t;
```

The structure fields have the following definitions:

sn_flag This type represents a single name component, and contains one of the following values:

```
SN_null             0
SN_typed            6
SN_objectid         7
SN_cds              8
SN_cdswildcard      9
```

The values 1, 2, 3, 4 and 5 are reserved.

SN_null Signifies an empty atomic name.

Note that empty components of external atomic names are not allowed, but this value may be used for protocol internal purposes.

SN_typed Signifies an atomic name representing a typed name (that is, an X.500 relative distinguished name (RDN)).

SN_objectid Signifies an attribute name.

SN_cds Signifies an atomic name representing an entity in the cell name space that does not contain wildcard metacharacters.

SN_cdswildcard Signifies an atomic name representing an entity in the cell name space that contains at least one non-escaped (that is, with no preceding \ (backslash)) wildcard metacharacter (that is, * (asterisk) or ? (question mark)).

sn_length Contains the length of the string contained in **sn_name**. If the value in **sn_flag** is **SN_null** (null opaque name), **sn_length** is zero.

sn_name Contains the string representation of the opaque name in question (with no terminating NULL). If the string represents an atomic name (that is, consists of one / (slash) separated name component), its syntax and canonical form are as specified in Section 9.1 on page 87.

If the value of **sn_flag** is **SN_null** (null opaque name), this field contains a null string.

The encoding of **sn_name** uses ISO 8859-1 code page (Latin-1) in ASCII.

The **FullName_t** structure defines an internal, opaque encoding of a complete global name to be passed through the CDS protocol. This structure is defined in **dns_record.h** as follows:

```
typedef uuid_t       ObjUID_t;
typedef bytes_u      ObjUID_u[sizeof(uuid_t)];

typedef struct {
   ObjUID_u   fn_root;
   word_u     fn_length;
   bytes_u    fn_name[FULLNAMEMAX - (sizeof(word_u) + sizeof(ObjUID_u))];
} FullName_t;
```

The structure fields have the following definitions:

fn_root Contains the UUID of the parent node of the composite name stored in **fn_name**. If an invalid name was processed (that is, there is no name in **fn_name**), then this field contains a NIL UUID.

fn_length Contains the total number of bytes in structures currently stored in **fn_name**.

fn_name Contains an array of **SimpleName_t** structures forming a complete global name.

11.2.2 Single Name Components

The structure representing the internal format of single name components is defined as follows (this structure is also used to contain attribute identifiers and class values):

```
typedef struct {
        unsigned short int              nm_length;
        [length_is(nm_length)] byte     nm_name[257];
        } cds_Name_t;
```

The structure fields have the following definitions:

nm_length Contains the length of the string in **nm_name**.

nm_name Contains an internal opaque name.

11.2.3 Full Names

The structure containing the string representation of the fully qualified global name is defined as follows:

```
typedef struct {
        uuid_t                         fn_root;
        long int                       fn_length;
        [length_is(fn_length)] char    fn_name[1023];
} cds_FullName_t;
```

The structure fields have the following definitions:

fn_root	Contains the UUID of the parent node (name context) of the composite name stored in **fn_name**. If an invalid name was processed, this field contains a NIL UUID.
fn_length	The length of the **fn_name** string.
fn_name	The canonical string representation of the name in question, relative to the context defined in **fn_root**. This is generally the global root (that is, the name includes the **/...** prefix). The server-to-server operations may use a UUID in **fn_root** that defines a CDS directory. The string does not include a terminating null. Its syntax and canonical form are specified in Section 1.3.2 on page 13.

11.3 Encoding of Time

11.3.1 CDS Timestamps

The CDS timestamp structure is defined as follows:

```
typedef struct {
        byte                  ts_node[6];
        unsigned hyper int    ts_time;
        } cds_Timestamp_t;
```

The structure fields have the following definitions:

ts_node The node identifier, containing an IEEE 802 address.

ts_time The amount of time, expressed in 100 nanosecond units, elapsed since 00:00:00.00, 17 November 1858 (Smithsonian calendar).

Note that the UTC base timestamps provided by the DCE Distributed Time Service are expressed as the amount of time elapsed since 15 October 1582 (Gregorian calendar).

11.3.2 Timeout Structure

The timeout structure used in operations for soft link entries is defined as follows:

```
typedef struct  {
        byte    to_expire[16];
        byte    to_extend[16];
        } cds_Timeout_t;

typedef [ptr]cds_Timeout_t *cds_TimeoutP_t;
```

The structure fields have the following definitions:

to_expire The absolute time determining the expiration of a soft link, encoded as Coordinated Universal Time (UTC), as specified in the **DCE Time Services** specification.

to_extend The relative time used as extension factor, encoded as UTC.

11.4 Encoding for Operations on Attributes

11.4.1 Atomic Attribute Values

All data types recognised by CDS are encoded in the discriminated union **cds_AtomicValue_t**, which is defined as follows:

```
typedef union switch (ValueType_t av_valuetype) av_val
  {
  case VT_none:            ;
  case VT_long:            long int                     av_long;
  case VT_short:           short int                    av_short;
  case VT_small:           small int                    av_small;
  case VT_uuid:            uuid_t                       av_uuid;
  case VT_Timestamp:       cds_Timestamp_t              av_timestamp;
  case VT_Timeout:         cds_Timeout_t                av_timeout;
  case VT_Version:         cds_Version_t                av_version;
  case VT_char:            [ptr]cds_OpenChar_t          *av_char_p;
  case VT_ASN1:
  case VT_byte:            [ptr]cds_OpenByte_t          *av_byte_p;

/* The remaining types are for internal CDS use only */

  case VT_ReplicaPointer:  [ptr]cds_ReplicaPointer_t   *av_rp_p;
  case VT_GroupMember:     [ptr]cds_GroupMember_t      *av_gm_p;
  case VT_ParentPointer:   [ptr]cds_ParentPointer_t    *av_pp_p;
  case VT_FullName:        [ptr]cds_FullName_t         *av_fullname_p;
  case VT_CHDirectory:     [ptr]cds_CHDirectory_t      *av_cp_p;
  case VT_DACL:            [ptr]sec_acl_t              *av_acl_p;
  case VT_gdaPointer:      [ptr]cds_gdaPointer_t       *av_gda_p;
  } cds_AtomicValue_t;
```

The structure fields have the following definitions:

VT_none	This indicates the absence of a value.
VT_long	This signifies a signed 32-bit integer.
VT_short	This signifies a signed 16-bit integer.
VT_small	This signifies a signed 8-bit integer.
VT_uuid	This signifies a universal unique identifier (UUID).
VT_Timestamp	This signifies a CDS timestamp (unique identifier); see above for encodings.
VT_Timeout	This signifies a timeout structure; see above for encodings.
VT_Version	This signifies class and replica version; see below for encodings.
VT_char	This signifies a counted string of 8-bit ASCII or EBCDIC characters (see Appendix D for encodings).
VT_ASN1	This signifies the same encoding as **VT_byte**, but representing ASN.1.
VT_byte	This signifies a counted string of untranslated octets (see Appendix D for encodings).
VT_ReplicaPointer	This signifies a replica pointer structure; see below for encodings.

VT_GroupMember	This signifies a group member structure (see Appendix D for encodings).
VT_ParentPointer	This signifies a parent pointer structure (see Appendix D for encodings).
VT_FullName	This signifies a full name structure; see above for encodings.
VT_CHDirectory	This signifies a clearinghouse directory structure (see Appendix D for encodings).
VT_DACL	This signifies an Access Control List structure (see the **DCE Security Services** specification for encodings).
VT_gdaPointer	This signifies a GDA pointer structure (see Appendix D for encodings).

The specified attribute type codes are as follows:

```
AT_none         1       /* no value */
AT_single       2       /* single-valued attribute */
AT_set          3       /* set-valued attribute */
```

11.4.2 Members of Set-Valued Attributes

The structure defining a member of a set-valued attribute or the contents of a single-valued attribute is defined as follows:

```
typedef struct {
        unsigned small int      sm_flag;
        cds_Timestamp_t         sm_ts;
        cds_AtomicValue_t       sm_value;
        } cds_SetMember_t;
```

The structure fields have the following definitions:

sm_flag The value of this flag indicates whether an attribute value is present or absent. Valid values are as follows:

```
SM_present      1
SM_absent       0
```

sm_ts Contains the member creation timestamp.

sm_value Contains the actual value of this attribute member.

11.4.3 Single and Set-Valued Attributes

The structure describing single- or set-valued attributes is defined as follows:

```
typedef struct {
        unsigned small int                      set_type;
        unsigned short int                      set_length;
        [size_is(set_length)] cds_SetMember_t   set_members[];
} cds_Set_t;
```

```
typedef [ptr]cds_Set_t *cds_SetP_t;
```

The structure fields have the following definitions:

set_type This flag describes the attribute type, and has one of the following values:

```
AT_none         1       /* no value */
AT_single       2       /* single-valued attribute */
AT_set          3       /* set-valued attribute */
```

set_length Contains the number of elements in the array **set_members**.

set_members An array containing the attribute's contents.

11.4.4 Values for Read Operations

The union that contains the value of the read attribute operation is defined as follows:

```
typedef union switch (unsigned small returningToClerk) {
    case RA_none:           ;
    case RA_single:         [ptr] cds_SetMember_t  *value_single_p;
    case RA_set:            [ptr] cds_Set_t        *value_set_p;
    case RA_wholeSet:       [ptr] cds_WholeEntry_t *wholeEntry_p;
} cds_RA_value_t;
```

11.4.5 Values for Modify Operations

The structure containing pertinent values for modify attribute operations is defined as follows:

```
typedef struct {
        small int               ud_operation;
        cds_Timestamp_t         ud_timestamp;
        unsigned small int      ud_type;
        byte                    ud_attribute[33];
        cds_AtomicValue_t       ud_value;
        } cds_Update_t;
```

The structure fields have the following definitions:

ud_operation A flag indicating whether the attribute is present or absent; its value is one of the following:

```
UD_present      1
UD_absent       2
```

ud_timestamp Contains the optional modification timestamp.

ud_type A flag describing the attribute type; its value is one of the following:

```
AT_none      1    /* used to delete an object entry */
AT_single    2
AT_set       3
```

ud_attribute A string representing the attribute identifier, encoded as a **SimpleName_t** structure with the value of its **sn_flag** being **SN_objectid**.

The attribute identifier represents the CCITT Object Identifier (OID) of the attribute, encoded in ASN.1/BER (ISO 8825). The directory version specified in this document supports only this format.

The attribute name to identifier mapping is specified in the CDS attributes table (in Appendix B).

ud_value Contains the attribute value.

11.5 Encoding of Progress Record

The structure specifying a progress record is defined as follows:

```
typedef struct {
        unsigned small int      pr_flags;
        [ptr]cds_Timeout_t      *pr_timeout;
        cds_FullName_t          pr_unresolved;
        cds_FullName_t          pr_resolved;
        [ptr]cds_Set_t          *pr_replicas_p;
        } cds_Progress_t;
```

The structure fields have the following definitions:

pr_flags A flag whose valid values are subsets (logical OR) of the following:

```
PR_done         0x00000001
PR_up           0x00000010
PR_linked       0x00000100
PR_hitLink      0x00001000
PR_ignoreState  0x00010000
PR_directory    0x00100000
```

pr_timeout A pointer to a value specifying the minimum timeout for all soft links followed in resolving a name.

pr_unresolved Contains the unresolved portion of the name.

pr_resolved Contains the resolved portion of the name.

pr_replicas_p A pointer to a field containing a set of clearinghouses.

11.6 Encoding of Replica Pointer

The internal CDS structure that describes replicas is defined as follows:

```
typedef struct {
        unsigned small int              rp_type;
        uuid_t                          rp_chid;
        cds_FullName_t                  rp_chname;
        unsigned long int               rp_length;
        [size_is(rp_length)] byte       rp_towers[];
        } cds_ReplicaPointer_t;

typedef [ref]cds_ReplicaPointer_t *cds_ReplicaPointerP_t;
```

The structure fields have the following definitions:

rp_type A flag specifying the type of replica; it can have one of the following values:

```
RT_master       1
RT_readOnly     3
RT_gda          4
```

The value 2 is reserved.

This document further specifies the usage only of **RT_master** (the Master Replica) and **RT_gda** (the Global Directory Agent).

rp_chid Contains the UUID of this replica.

rp_chname Contains the string representation of the name of this replica.

rp_length Contains the number of elements of the **rp_towers** array.

rp_towers An array containing the set of protocol towers for this replica (see the **DCE Remote Procedure Call** specification for encodings).

11.7 Miscellaneous Data Types and Constants

11.7.1 Boolean Values

Any integral or pointer type that is used to represent Boolean values is encoded as follows:

FALSE	equal to zero
TRUE	unequal to zero.

11.7.2 Entry Type

The structure that specifies entry type is defined as follows:

```
typedef struct {
        byte_u           et_value;
        } EntryType_t;
```

The structure fields have the following definitions:

et_value A flag that specifies the entry type; it has one of the following values:

```
ET_directory            1
ET_object               2
ET_childPointer         3
ET_softlink             4
ET_clearinghouse        5
ET_anyDirectName        6
ET_firstLink            7
ET_dirOrObj             8
```

11.7.3 Clearinghouse List

The structure that specifies a list of clearinghouses is defined as follows:

```
typedef struct {
    unsigned short                              ch_length;
    [size_is(ch_length)] cds_ReplicaPointerP_t   ch_members[];
    } cds_CH_t;
```

The structure fields have the following definitions:

ch_length Contains the number of elements in the array **ch_members**.

ch_members The array of replica pointers.

11.7.4 Clearinghouse State

The structure that determines the state of a clearinghouse is defined as follows:

```
typedef struct {
        byte_u  cs_value;
} CHState_t;
```

The structure field has the following definition:

cs_value A flag that specifies the state, which can be one of the following:

```
CS_newCH        1
CS_on           2
CS_dyingCH      3
```

11.7.5 Version Number

The structure defining a version number is defined as follows:

```
typedef struct {
        unsigned small int    ver_major;
        unsigned small int    ver_minor;
        } cds_Version_t;
```

The structure fields have the following values:

ver_major Contains the major version number.

ver_minor Contains the minor version number.

11.7.6 Principal and Group Identities

The **sec_id_t** structure is the basic unit for identifying principals or groups. It is defined as follows:

```
typedef struct {
        uuid_t                  uuid;
        [ string,ptr ] char     *name;
        } sec_id_t;
```

The structure fields have the following definitions:

uuid Contains the UUID that provides an object handle for the identity.

name Contains the optional printstring name.

11.7.7 Foreign Identities

The **sec_id_foreign_t** structure defines an identity from a foreign realm. It is defined as follows:

```
typedef struct {
        sec_id_t            id;
        sec_id_t            realm;
        } sec_id_foreign_t;
```

The structure fields have the following definitions:

id Contains the identifier of the foreign user or group.

realm Contains the identifier of the foreign realm.

11.7.8 Error Status Returns

The structure for an error status return is defined as follows:

```
typedef struct {
        unsigned long int       er_status;
        [ptr] cds_FullName_t    *er_name;
        } cds_status_t;
```

The structure fields have the following definitions:

er_status	Contains the status number.
er_name	Contains a pointer to a full name if the status [CDS_UNKNOWN_ENTRY] is returned.

Chapter 12

IDL Notation of CDS Operations

This chapter contains specifications for the RPC interfaces used by CDS clerks and servers.

The RPC interfaces are specified in IDL (Interface Definition Language). Part of the information in these IDL declarations is symbolic and need not be preserved identically. For example, the names of procedures and parameters are a local matter. The information that must be identical for a conforming implementation is as follows:

- the interface identifier, composed of UUID and version
- the order of procedures within the interface definition (see Appendix D)
- the order and number of parameters in procedure signatures
- the types of parameters and procedures
- the attributes of parameters and procedures.

The following sections contain the interface specifications for CDS remote operations.

12.1 cds_Advertise( )

```
[broadcast,maybe] void cds_Advertise(
        [in] handle_t        h,
        [in] cds_FullName_t  *cellname_p,
        [in] uuid_t          cell_diruid,
        [in] cds_CH_t        *nscle_p
        );
```

cds_Advertise( ) advertises the CDS server's availability and the availability of each of its running clearinghouses. CDS servers send advertisements when one of the following is true:

- The advertisement timer expires.
- A solicitation request (operation *cds_Solicit*( )) is received.

The advertisement intervals for CDS server advertisements are implementation-dependent. A recommended default for advertisement intervals is five minutes. Implementations should apply jitter to this timeout value within a cell name space that is running replicated servers to avoid synchronised broadcasts.

No permission is required for the operation.

Input Parameters

The input parameters for *cds_Advertise*( ) are:

h	RPC binding handle.
cellname_p	The fully-qualified global name of the local cell.
cell_diruid	The UUID of the cell.
nscle_p	The list of available clearinghouses at the server, including its UUIDs, fully-qualified global names, and exported protocol towers. The **rp_type** flag of this data structure determines the replica type.

12.2 cds_CreateChild( )

```
error_status_t cds_CreateChild(
        [in] handle_t            h,
        [in,out] cds_Progress_t   *Progress_p,
        [in,ptr] sec_id_foreign_t *user_p,
        [in] uuid_t              *childID_p,
        [in] cds_Set_t           *replicaset_p,
        [out] uuid_t             *parentID_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_CreateChild( ) creates a child pointer entry in the parent directory.

This operation is performed only in server-to-server communication as a result of a *cds_CreateDirectory*( ) operation which may affect a parent directory located in a different clearinghouse on a remote server.

The operation requires **insert** access permission to the parent directory.

Input Parameters

The input parameters for *cds_CreateChild*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
user_p	Identity of the principal that is invoking this operation. This client's EPAC is used at the target server to verify the initiator's identity and access permissions (requesting server impersonates the initiator), for operations that set up an Access Control List on the child pointer entry. The appropriate security service operations are assumed (refer to the **DCE Security Services** specification).
childID_p	The UUID of the child.
replicaset_p	Replica set. Each member of the set contains one replica pointer (the **sm_value** fields are of type **VT_ReplicaPointer**).

Input/Output Parameters

The input/output parameters for *cds_CreateChild*( ) are:

Progress_p	Progress record, used at the server's clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the child entry created.

Output Parameters

The output parameters for *cds_CreateChild*( ) are:

parentID_p	The UUID of the parent directory.
cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_ENTRYEXISTS]

12.3 cds_CreateDirectory( )

```
error_status_t cds_CreateDirectory(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [out] cds_Timestamp_t    *actual_ts_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_CreateDirectory( ) creates a directory entry as a child of the directory entry implied by the fully-qualified global name of the new directory.

The operation requires **insert** access permission to the parent directory, and **write** access permission to the clearinghouse which is to store the master replica for the directory.

Note also that this operation performs a *cds_CreateChild*( ) operation and therefore the server requires respective access permissions to its peer server.

Input Parameters

The input parameters for *cds_CreateDirectory*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.

Input/Output Parameters

The input/output parameters for *cds_CreateDirectory*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the directory entry created.

Output Parameters

The output parameters for *cds_CreateDirectory*( ) are:

actual_ts_p	A timestamp indicating the date and time that the directory was created.
cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_CANTPUTHERE]
[CDS_ENTRYEXISTS]

12.4 cds_CreateObject( )

```
error_status_t cds_CreateObject(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [in,ptr] cds_Name_t      *class_p,
        [in,ptr] cds_Version_t   *version_p,
        [in, ptr] uuid_t         *uuid_p,
        [out] cds_Timestamp_t    *actual_ts_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_CreateObject( ) creates an object entry in the cell name space. Creating an object entry requires meeting the following conditions:

- The name given for the object entry must be new for the cell name space. If a directory, object entry, clearinghouse or soft link already exists with the specified name, the operation is unsuccessful.
- All directories cited in the fully-qualified global name must exist.

The entry initially has the **CDS_CTS** and **CDS_UTS** attributes, and optionally, the **CDS_Class**, **CDS_ClassVersion** and **CDS_ObjectUUID** attributes. Other attributes may be added via the *cds_ModifyAttribute*( ) operation.

The operation requires **insert** access permission to the parent directory.

Input Parameters

The input parameters for *cds_CreateObject*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
class_p	A printable string name representing the object's class. The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to either **SN_cds** or **SN_cdswildcard**. This parameter is optional. If it contains a non-NULL value, *version_p* is also used.
version_p	The major and minor class version numbers of the application. This parameter is used only if *class_p* is non-NULL.
uuid_p	The optional unique identifier for the object entry.

Input/Output Parameters

The input/output parameters for *cds_CreateObject*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the object entry created.

Output Parameters

The output parameters for *cds_CreateObject*( ) are:

actual_ts_p	A timestamp indicating the date and time the object entry was created.
cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_ENTRYEXISTS]

12.5 cds_CreateSoftLink( )

```
error_status_t cds_CreateSoftLink(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [in] cds_FullName_t      *target_p,
        [in,ptr] cds_Timeout_t   *linkTimeout_p,
        [out] cds_Timestamp_t    *actual_ts_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_CreateSoftLink( ) creates a soft link entry with the name as identified in the progress record that points to another entry whose name is identified by *target_p*.

The operation requires **insert** access permission to the directory in which the soft link is created.

Input Parameters

The input parameters for *cds_CreateSoftLink*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
target_p	The string representation of the fully-qualified global name of an existing entry in the cell name space to which the soft link (identified in *Progress_p*) points.
linkTimeout_p	A structure that specifies two values: the first value is an absolute time after the elapse of which the soft link is deleted if its target entry no longer exists; the second value is an extension factor for the timeout. When the time determined by the first value expires, the CDS server attempts to verify that the entry to which the link points still exists, and extends the life of the soft link by the second value only if it does still exist. If this operation parameter is set to NULL, the soft link is neither checked nor deleted by the CDS server on the client's behalf.

Input/Output Parameters

The input/output parameters for *cds_CreateSoftLink*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the soft link to be created.

Output Parameters

The output parameters for *cds_CreateSoftLink*( ) are:

actual_ts_p	A timestamp indicating the date and time the soft link was created.
cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_ENTRYEXISTS]

12.6 cds_DeleteChild( )

```
error_status_t cds_DeleteChild(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [out] cds_status_t      *cds_status_p
        );
```

cds_DeleteChild( ) deletes the child pointer entry from the parent directory as part of deleting a directory from the cell name space.

This operation is only performed in server-to-server communication as a result of a *cds_DeleteDirectory*( ) operation if the parent directory is located in a different clearinghouse on a remote server.

The operation requires either **delete** access permission to the child pointer, or **administer** access permission to the parent directory entry.

Input Parameters

The input parameters for *cds_DeleteChild*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.

Input/Output Parameters

The input/output parameters for *cds_DeleteChild*( ) are:

Progress_p	Progress record, used at the server's clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the child entry deleted.

Output Parameters

The output parameters for *cds_DeleteChild*( ) are:

cds_status_p Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]

12.7 cds_DeleteDirectory( )

```
error_status_t cds_DeleteDirectory(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [out] cds_status_t      *cds_status_p
        );
```

cds_DeleteDirectory( ) removes the specified directory entry from the cell name space. The directory must be empty in order to be deleted.

The operation requires **delete** access permission to the directory entry, and **write** permission to the clearinghouse that stores the master replica.

Note also that this operation performs a *cds_DeleteChild*( ) operation, and therefore the server requires respective access permissions to its peer server.

Input Parameters

The input parameters for *cds_DeleteDirectory*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.

Input/Output Parameters

The input/output parameters for *cds_DeleteDirectory*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the directory entry to be deleted.

Output Parameters

The output parameters for *cds_DeleteDirectory*( ) are:

cds_status_p Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_NOTEMPTY]

12.8 cds_DeleteObject( )

```
error_status_t cds_DeleteObject(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [out] cds_status_t      *cds_status_p
        );
```

cds_DeleteObject( ) removes the specified object entry from the cell name space.

The operation requires **delete** access permission to the object entry, or **administer** permission to its parent directory.

Input Parameters

The input parameters for *cds_DeleteObject*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.

Input/Output Parameters

The input/output parameters for *cds_DeleteObject*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the object entry to be deleted.

Output Parameters

The output parameters for *cds_DeleteObject*( ) are:

cds_status_p Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]

12.9 cds_DeleteSoftLink( )

```
error_status_t cds_DeleteSoftLink(
        [in] handle_t            h,
        [in,out] cds_Progress_t  *Progress_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_DeleteSoftLink( ) deletes a soft link from the cell name space.

The operation requires **delete** access permission to the soft link, or **administer** permission to its parent directory.

Input Parameters

The input parameters for *cds_DeleteSoftLink*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.

Input/Output Parameters

The input/output parameters for *cds_DeleteSoftLink*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the soft link to be deleted.

Output Parameters

The output parameters for *cds_DeleteSoftLink*( ) are:

cds_status_p Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]

12.10 cds_EnumerateAttributes( )

```
[idempotent] error_status_t cds_EnumerateAttributes(
        [in] handle_t             h,
        [in,out] cds_Progress_t   *Progress_p,
        [in] unsigned small       type,
        [in] cds_Name_t           *context_p,
        [in] unsigned32           max_size,
        [in,out,ptr] cds_SetP_t   *attr_set,
        [out] unsigned small      *wholeset_p,
        [out] cds_status_t        *cds_status_p
        );
```

cds_EnumerateAttributes( ) returns in **attr_set** a set whose members are the attribute identifiers of each currently present attribute of the entry whose type is specified in **type**.

Note: The attribute identifiers are enumerated in lexical order of the string representation of object identifiers (CCITT OIDs). If the call returns FALSE (parameter *wholeset_p*), not all attributes have been enumerated and the client may make further calls, setting *context_p* to the last attribute in the set returned, until the operation returns TRUE.

The operation requires **read** access permission to the entry whose attributes are to be enumerated.

Input Parameters

The input parameters for *cds_EnumerateAttributes*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
type	The type of the entry for which attributes are enumerated. Valid entry types are: **ET_directory** **ET_object** **ET_childPointer** **ET_softlink** **ET_clearinghouse** **ET_dirOrObj**
context_p	The context of the last attribute in the previous call (the attribute identifier returned in **attr_set**). The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to **SN_objectid**. This is used in a sequence of calls to enumerate the whole set of attributes of this entry. If set to NULL, the look-up operation starts at the beginning of the entry.
max_size	The maximum size of values to be returned by **attr_set** per call instance.

Input/Output Parameters

The input/output parameters for *cds_EnumerateAttributes*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the entry for which attributes are enumerated.
attr_set	A pointer to the set of the requested attribute identifiers (the attribute name to identifier mapping is specified in Section B.2 on page 186). These attribute identifiers are CCITT object identifiers, encoded in ASN.1, BER. The attribute

identifiers are contained in the **sm_value** fields (type **VT_byte**), encoded as **SimpleName_t** structures. Attribute values are absent.

The buffer that *attr_set* points to must not exceed **max_size**.

Output Parameters

The output parameters for *cds_EnumerateAttributes*( ) are:

wholeset_p	A boolean value that indicates whether this call instance returned the whole set of enumerated attributes.
cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_NOTLINKED]
[CDS_DANGLINGLINK]

12.11 cds_EnumerateChildren()

```
[idempotent] error_status_t cds_EnumerateChildren(
        [in] handle_t            h,
        [in,out] cds_Progress_t   *Progress_p,
        [in] cds_Name_t          *wild_p,
        [in] cds_Name_t          *context_p,
        [in] unsigned32          max_size,
        [in,out,ptr] cds_SetP_t   *name_set,
        [out] unsigned small      *wholeset_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_EnumerateChildren( ) takes as input the name of a directory and a wildcarded component name to match against child pointer entries in the directory. It returns a set whose members are the component names of child directories of the directory which matched the wildcarded name. If no children matched the wildcard or the directory has no children, a NULL set is returned.

The child pointers are enumerated in lexical order (based on the Portable Character Set (PCS)). If the call returns FALSE (parameter *wholeset_p*), not all children have been enumerated and the client may make further calls, setting *context_p* to the last child directory name in the set returned, until the operation returns TRUE.

This operation requires **read** access permission to the parent directory and to each child pointer.

Input Parameters

The input parameters for *cds_EnumerateChildren*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
wild_p	A wildcard filter for the child directory names. The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to either **SN_cds** or **SN_cdswildcard**. This operation parameter is optional, and can be set to NULL. Specifying a NULL value is the same as specifying the string * (asterisk character).
context_p	The context of the last child directory in the previous call (the child directory name returned in *name_set*). The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to **SN_cds**. This is used in a sequence of calls to enumerate the whole set of child directories of this entry. If set to NULL, the look-up operation starts at the beginning of the entry.
max_size	The maximum size of values to be returned by *name_set* per call instance.

Input/Output Parameters

The input/output parameters for *cds_EnumerateChildren*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the directory for which the child directories are enumerated.
name_set	A pointer to the set of the requested child directories. The child directory names are contained in the **sm_value** fields (type **VT_byte**), encoded as a **SimpleName_t** structure. The buffer that *name_set* points to must not exceed **max_size**.

Output Parameters

The output parameters for *cds_EnumerateChildren*( ) are:

wholeset_p	A boolean value that indicates whether this call instance returned the whole set of enumerated child directory entries.
cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]

12.12 cds_EnumerateObjects( )

```
[idempotent] error_status_t cds_EnumerateObjects(
        [in] handle_t            h,
        [in,out] cds_Progress_t   *Progress_p,
        [in] cds_Name_t          *wild_p,
        [in] cds_Name_t          *context_p,
        [in] cds_Name_t          *class_p,
        [in] unsigned32          max_size,
        [in,out,ptr] cds_SetP_t   *name_set,
        [out] unsigned small      *wholeset_p,
        [in,out] unsigned small   *returnClass_p,
        [out] cds_status_t        *cds_status_p
        );
```

cds_EnumerateObjects( ) takes as input the name of a directory, a wildcarded component name, and optionally a filter on object class to match against object entries in the directory. It returns a set whose members are either the component names of object entries in the directory which matched the wildcarded name, or the object names paired with the object class, depending on the value of the *returnClass_p* parameter. If no object entries matched the wildcard or the directory contains no object entries, a NULL set is returned.

The objects are enumerated in lexical order (based on the Portable Character Set (PCS)). If the call returns FALSE (parameter *wholeset_p*), not all matching objects have been enumerated, and the client may make further calls, setting *context_p* to the last object name in the set returned, until the operation returns TRUE.

If the filter *class_p* is specified, only those objects of the specified class are returned. The filter may be a wildcard class.

This operation requires **read** access permission to the directory and objects.

Input Parameters

The input parameters for *cds_EnumerateObjects*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
wild_p	A wildcard filter for object entry names in the specified directory. The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to either **SN_cds** or **SN_cdswildcard**. This operation parameter is optional and can be set to NULL. Specifying a NULL value is the same as specifying the string * (asterisk character).
context_p	The context of the last object entry name in the previous call (the object entry name returned in *name_set*). The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to **SN_cds**. This is used in a sequence of calls to enumerate the whole set of object entries of the directory. If set to NULL, the look-up operation starts at the beginning.
class_p	A filter for class names (may be wildcarded) of object entries that use the **CDS_Class** attribute. The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to either **SN_cds** or **SN_cdswildcard**. This operation parameter is optional and can be set to NULL. Specifying a NULL value is the same as specifying the string * (asterisk character).
max_size	The maximum size of values to be returned by *name_set* per call instance.

Input/Output Parameters

The input/output parameters for *cds_EnumerateObjects*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the directory for which the object entries are enumerated.
name_set	A pointer to the set of the requested object entry names. Object and class names are contained in the **sm_value** fields (type **VT_byte**), encoded as a **SimpleName_t** structure. If *returnClass_p* is TRUE, each member of the returned set contains in the **sm_value** field a sequence of object entry name immediately followed by class name (encoded as **SimpleName_t**). The buffer that *name_set* points to must not exceed **max_size**.
returnClass_p	Indicates whether the object's class is returned. If set to TRUE, the object's class is returned along with the object name. If set to FALSE, the object's class is not returned.

Output Parameters

The output parameters for *cds_EnumerateObjects*( ) are:

wholeset_p A boolean value that indicates whether this call instance returned the whole set of enumerated object entries.

cds_status_p Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]

12.13 cds_EnumerateSoftLinks( )

```
[idempotent] error_status_t cds_EnumerateSoftLinks(
        [in] handle_t             h,
        [in,out] cds_Progress_t   *Progress_p,
        [in] cds_Name_t           *wild_p,
        [in] cds_Name_t           *context_p,
        [in] unsigned32           max_size,
        [in,out,ptr] cds_SetP_t   *name_set,
        [out] unsigned small      *wholeset_p,
        [out] cds_status_t        *cds_status_p
        );
```

cds_EnumerateSoftLinks( ) takes as input the name of a directory and a wildcarded component name to match against soft link entries in the directory. It returns a set whose members are the component names of soft link entries in the directory which matched the wildcarded name. If no soft link entries matched the wildcard or the directory contains no soft link entries, a NULL set is returned.

The soft links are enumerated in lexical order (based on the Portable Character Set (PCS)). If the call returns FALSE (parameter *wholeset_p*), not all matching soft links have been enumerated and the client may make further calls, setting *context_p* to the last soft link in the set returned, until the operation returns TRUE.

This operation requires **read** access permission to the directory and soft links.

Input Parameters

The input parameters for *cds_EnumerateSoftLinks*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
wild_p	A wildcard filter for the soft links in the specified directory. The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to either **SN_cds** or **SN_cdswildcard**. This operation parameter is optional and can be set to NULL. Specifying a NULL value is the same as specifying the string * (asterisk character).
context_p	The context of the last soft link in the previous call (the soft link name returned in *name_set*). The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to **SN_cds**. This is used in a sequence of calls to enumerate the whole set of soft link entries of the directory. If set to NULL, the look-up operation starts at the beginning.
max_size	The maximum size of values to be returned by *name_set* per call instance.

Input/Output Parameters

The input/output parameters for *cds_EnumerateSoftLinks*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the directory for which soft links are enumerated.

name_set A pointer to the set of the requested soft link values. The soft link entry names are contained in the **sm_value** fields (type **VT_byte**), encoded as **SimpleName_t** structures. The buffer that *name_set* points to must not exceed **max_size**.

Output Parameters

The output parameters for *cds_EnumerateSoftLinks*( ) are:

wholeset_p A boolean value that indicates whether this call instance returned the whole set of enumerated soft links.

cds_status_p Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]

12.14 cds_ModifyAttribute( )

```
[idempotent] error_status_t cds_ModifyAttribute(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [in] unsigned small     type,
        [in] cds_Update_t       *update_p,
        [out] cds_status_t      *cds_status_p
        );
```

cds_ModifyAttribute( ) applies one update to the specified entry. An attribute may be removed or added, or a value may be removed or added to the values of a set-valued attribute, or the value of a single-valued attribute may be altered. This operation may also remove an entire object entry from the name space.

If the target entry is a directory, only certain attributes of the directory may be directly modified with *cds_ModifyAttribute*( ).

This operation requires either **write** access permission to the entry whose attribute is being modified (if the operation makes the attribute present or absent), or **write** access permission to the parent directory.

Input Parameters

The input parameters for *cds_ModifyAttribute*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
type	The type of the entry for which attributes are modified. Valid entry types are: **ET_directory** **ET_object** **ET_childPointer** **ET_softlink** **ET_clearinghouse**
update_p	The structure defining the update operation, type, identifier, value and timestamp of the attribute to be modified.

Valid values for the update operation flag are:

UD_present	Make attribute or value (or both) present; that is, either add attribute or alternative value.
UD_absent	Make object entry, attribute or attribute value absent; that is, remove it.

Valid values for the attribute type are:

AT_none
AT_single
AT_set

The result of this operation is determined by the combination of the three fields defining operation, type and value. The relationship is listed in Table 12-1 on page 159.

Operation	Attribute		Result
	Type	**Value**	
UD_present	AT_none	-	Invalid: returns status [CDS_WRONGATTRIBUTETYPE].
	AT_single	value	Creates new attribute or replaces value if attribute exists.
	AT_single	empty	Invalid: returns status [CDS_INVALIDUPDATE].
	AT_set	value	Creates new attribute or adds value to existing attribute.
	AT_set	empty	Creates new attribute (empty set). Invalid if attribute exists (returns status [CDS_INVALIDUPDATE]).
UD_absent	AT_none	ignored	Deletes object entry (invalid on other entry types).
	AT_single	ignored	Removes attribute from entry.
	AT_set	any value (except **VT_none**)	Removes value from attribute.
	AT_set	value type **VT_none**	Removes attribute from entry.

Table 12-1 Modify Operations

The *update_p* data structure contains a field in which the client may supply a timestamp to be used for the update. If the client chooses not to supply its own timestamp, this field (**ud_timestamp**) must be set to all zeros (**NullTimestamp**).

If the client chooses to supply a timestamp and the CDS server considers the timestamp invalid, the [CDS_BADCLOCK] failure status is returned.

Input/Output Parameters

The input/output parameters for *cds_ModifyAttribute*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the target entry of which the attribute is to be modified.

Output Parameters

The output parameters for *cds_ModifyAttribute*( ) are:

cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_UNKNOWNATTRIBUTE]
[CDS_WRONGATTRIBUTETYPE]
[CDS_NOTSUPPORTED]
[CDS_INVALIDUPDATE]
[CDS_BADCLOCK]

12.15 cds_ReadAttribute( )

```
[idempotent] error_status_t cds_ReadAttribute(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [in] unsigned small      type,
        [in] cds_Name_t          *att_p,
        [in] cds_Timestamp_t     *context_p,
        [in] unsigned32          max_size,
        [in] unsigned32          maybemore,
        [out] cds_RA_value_t     *value_p,
        [out] unsigned small     *wholeset_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_ReadAttribute( ) returns the values of the specified attribute. If *maybemore* is set to TRUE, the whole set of attributes contained in the specified entry may be returned (this behaviour is implementation-dependent).

The attribute values are returned in timestamp order (oldest timestamp first). If all values have not been returned and the client makes further calls, *context_p* must be set to the timestamp of the last attribute in the set returned.

The operation requires **read** access permission to the entry whose attribute value is to be read.

Input Parameters

The input parameters for *cds_ReadAttribute*( ) are:

h RPC binding handle referring to the target server and clearinghouse.

type The type of the entry for which attributes are read. Valid entry types are:

ET_directory
ET_object
ET_childPointer
ET_softlink
ET_clearinghouse
ET_dirOrObj

att_p The identifier of the attribute. The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to **SN_objectid**.

This attribute identifier may have been previously obtained through the *cds_EnumerateAttributes* operation.

context_p The context (timestamp) of the last attribute in the previous call (the timestamp **sm_ts** returned in *value_p*). This is used in a sequence of calls to read the whole set of set-valued attributes. If set to NULL, the look-up operation starts at the beginning of the specified attribute value set.

max_size The maximum size of values to be returned by *value_p* per call instance.

maybemore This is a boolean flag to indicate (if set to TRUE) that the client intends to read the attribute values of the whole entry, and requests optimised use of resources.

If the clerk sets the *maybemore* parameter to TRUE, the CDS server may do extra work to attempt to make subsequent *cds_ReadAttribute*( ) calls for the same entry more efficient. If the buffer determined by **max_size** is big enough, the server may return the attribute values of the whole entry in *value_p*. In this case, the parameters *att_p* and *context_p* are ignored.

The clerk may set this argument to TRUE on any call, but improved performance may result only if the clerk's cache is large enough to hold all the relevant information.

CDS server implementations may not support this behaviour and instead always return only the values of a maximum of one attribute.

Input/Output Parameters

The input/output parameters for *cds_ReadAttribute*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the entry whose attribute is read.

Output Parameters

The output parameters for *cds_ReadAttribute*( ) are:

value_p

A pointer to the set of the requested attribute values.

If the attribute is not present in the entry, the **sm_flag** of *value_p* has the value **SM_absent**, and the set contents are NULL.

An empty set-valued attribute is indicated by the **sm_flag** having the value **SM_present**, and the set contents being NULL.

The buffer that *value_p* points to must not exceed **max_size**.

wholeset_p

Not supported (reserved for future use).

cds_status_p

Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_NONSRESOURCES]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_UNKNOWNATTRIBUTE]
[CDS_WRONGATTRIBUTETYPE]
[CDS_NOTLINKED]
[CDS_DANGLINGLINK]
[CDS_NAMESERVERBUG]

12.16 cds_ResolveName( )

```
[idempotent] error_status_t cds_ResolveName(
        [in] handle_t           h,
        [in,out] cds_Progress_t  *Progress_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_ResolveName( ) follows a chain of soft links until it reaches an entry that is not a soft link. It returns the fully-qualified global name of that entry so that future calls by the client may use the direct name without incurring the overhead of following the link.

If the target of any of the chain of soft links followed does not exist, the [CDS_DANGLINGLINK] failure status is returned.

This operation requires **read** access permission to each of the soft links in the chain.

Input Parameters

The input parameters for *cds_ResolveName*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.

Input/Output Parameters

The input/output parameters for *cds_ResolveName*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the soft link to be resolved.

Output Parameters

The output parameters for *cds_ResolveName*( ) are:

cds_status_p Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_NOTLINKED]
[CDS_DANGLINGLINK]

12.17 cds_Solicit( )

```
[broadcast,maybe] void cds_Solicit(
        [in] handle_t  h
        );
```

cds_Solicit( ) is used to provoke advertisement and carries no information of its own. Therefore it has the very simple structure of just the operation header.

No permissions are required for the operation.

Input Parameters

The input parameters for *cds_Solicit*( ) are:

h	RPC binding handle referring to the target.

12.18 cds_SolicitServer( )

```
[idempotent] error_status_t cds_SolicitServer(
        [in] handle_t          h,
        [out] cds_FullName_t    *cellname_p,
        [out] uuid_t            *cell_diruid_p,
        [in,out,ptr] cds_CHP_t  *nscle_p
        );
```

cds_SolicitServer( ) is a directed solicitation request to obtain availability information about clearinghouses of a particular CDS server. The data structures in the response message (output parameters) are similar to those transmitted in the *cds_Advertise*( ) operation.

No permissions are required for the operation.

Input Parameters

The input parameters for *cds_SolicitServer*( ) are:

h	RPC binding handle referring to the target server.

Input/Output Parameters

The input/output parameters for *cds_SolicitServer*( ) are:

nscle_p	If provided as an input parameter, the information contained is a list of requested clearinghouses. The output parameter contains the list of available clearinghouses at the server, including its UUIDs, fully-qualified global name, and exported protocol towers. The **rp_type** flag of this data structure determines the replica type.

Output Parameters

The output parameters for *cds_SolicitServer*( ) are:

cellname_p	The fully-qualified global name of the local cell.
cell_diruid_p	The UUID of the cell name space.

12.19 cds_TestAttribute( )

```
[idempotent] error_status_t cds_TestAttribute(
        [in] handle_t            h,
        [in,out] cds_Progress_t  *Progress_p,
        [in] unsigned small      type,
        [in] cds_Name_t          *att_p,
        [in] cds_AtomicValue_t   *value_p,
        [out] unsigned small     *result_p,
        [out] cds_status_t       *cds_status_p
        );
```

cds_TestAttribute( ) returns a boolean value in *result_p* which is TRUE if one of the following is true:

- The specified attribute is single attribute and its value matches the client-specified value.
- The specified attribute is a set-valued attribute and it contains the client-specified value as one of its members.

If the attribute is not present in the entry, the operation returns FALSE.

The operation requires **read** or **test** access permission to the entry whose attribute is to be tested.

Input Parameters

The input parameters for *cds_TestAttribute*( ) are:

h	RPC binding handle referring to the target server and clearinghouse.
type	The type of the entry for which attributes are tested. Valid entry types are: **ET_directory** **ET_object** **ET_childPointer** **ET_softlink** **ET_clearinghouse** **ET_dirOrObj**
att_p	The identifier of the attribute. The **nm_name** field is encoded as a **SimpleName_t** structure with its **sn_flag** set to **SN_objectid**.
value_p	The attribute value to be tested. The data in this operation parameter is a discriminated union, consisting of a syntax identifier and value.

Input/Output Parameters

The input/output parameters for *cds_TestAttribute*( ) are:

Progress_p	Progress record, used at the clerk for handling referrals to another server. The **pr_unresolved** field of the progress record contains the portion of the original fully-qualified global name of the entry for which an attribute value is tested.

Output Parameters

The output parameters for *cds_TestAttribute*( ) are:

result_p	The boolean value indicating the result of the test operation.
cds_status_p	Error status return. On [CDS_UNKNOWNENTRY], the **er_name** field may be filled in with the last name the server successfully accessed.

Possible status codes are (non-exclusive):

[CDS_SUCCESS]
[CDS_VERSIONSKEW]
[CDS_ACCESSDENIED]
[CDS_CANNOTAUTHENTICATE]
[CDS_UNTRUSTEDCH]
[CDS_CLEARINGHOUSEDOWN]
[CDS_POSSIBLECYCLE]
[CDS_ROOTLOST]
[CDS_UNDERSPECIFIEDNAME]
[CDS_UNKNOWNENTRY]
[CDS_NOTLINKED]
[CDS_DANGLINGLINK]

Part 5

Appendices

The Open Group

Appendix A

Valid Characters and Naming Rules

This appendix summarises the valid character sets for DCE Directory Service names, defined in the Portable Character Set (PCS). It also explains some characters that have special meaning and describes some restrictions and rules regarding case matching, syntax and size limits.

The use of names in **X/Open DCE** often involves more than one directory service. For example, CDS interacts with either GDS or DNS to find names outside the local cell.

Note: Because CDS, GDS and DNS all have their own valid character sets and syntax rules, the best way to avoid problems is to keep names short and simple, consisting of a minimal set of characters common to all three services. The recommended set is the letters A to Z, a to z, and the digits 0 to 9. In addition to making directory service interoperations easier, use of this subset decreases the probability that users in a heterogeneous hardware and software environment encounter problems creating and using names.

Figure A-1 on page 172 details the valid characters in CDS names, and the valid characters in GDS and DNS names as used by CDS:

- Characters in white boxes are valid in all three kinds of names.
- Characters in light shaded boxes are valid only in CDS and GDS names.
- Characters in dark shaded boxes are valid only in CDS names.

Implementations may support additional national character sets to be used in names and attribute values. However, implementations that provide for support of additional character sets may lose interoperability to other implementations that conform to this specification.

SP ░	0	@ ▓	P	‘ ▓	p
! ▓	1	A	Q	a	q
“ ▓	2	B	R	b	r
# ▓	3	C	S	c	s
$ ▓	4	D	T	d	t
% ▓	5	E	U	e	u
& ▓	6	F	V	f	v
’ ░	7	G	W	g	w
(░	8	H	X	h	x
) ░	9	I	Y	i	y
* ▓	: ░	J	Z	j	z
+ ░	; ▓	K	[▓	k	{ ▓
, ░	< ▓	L	\ ░	l	| ▓
-	= ░	M	] ▓	m	} ▓
.	> ▓	N	^ ▓	n	~ ▓
/ ░	? ░	O	_ ▓	o	

Key:
- □ Valid in CDS, GDS and DNS names
- ░ Valid only in CDS and GDS names
- ▓ Valid only in CDS names

Figure A-1 Valid Characters in CDS, GDS and DNS Names

A.1 Valid Characters for GDS Naming Attributes

The values of the country attributes are restricted to the ISO 3166 Alpha-2 code representation of country names.

The character set for all other naming attributes is chosen by the user of the XDS interface. The character set is dependent on the syntax of the selected attribute:

- For printable strings, the valid characters are those that are displayed as valid characters for GDS as shown in Figure A-1 on page 172.
- The T.61 graphical character set is specified in Section A.3 on page 177.
- The ISO 8859-1 (Latin-1) graphical character set is specified in Section A.4 on page 178.
- For numeric strings, the valid character set is 0-9 and the space character.
- For the IA5 character set, all characters with a coded representation of lower than or equal to 0x7f are valid. The GDS restricts the valid characters to range from 0x07 to 0x7f inclusive.

The default schema of GDS contains only naming attributes that specify character sets of type Printable string or T.61 graphical character set.

A.2 Country Syntax

Country names are represented by a two-letter sequence. GDS does not distinguish between lower case and upper case for country names. The complete list of valid combinations is shown in Table A-1 together with the respective names.

Table A-1 Country Syntax

Country Name	Code	Country Name	Code
AFGHANISTAN	AF	ALBANIA	AL
ALGERIA	DZ	AMERICAN SAMOA	AS
ANDORRA	AD	ANGOLA	AO
ANGUILLA	AI	ANTARCTICA	AQ
ANTIGUA AND BARBUDA	AG		
ARGENTINA	AR	ARUBA	AW
AUSTRALIA	AU	AUSTRIA	AT
BAHAMAS	BS	BAHRAIN	BH
BANGLADESH	BD	BARBADOS	BB
BELGIUM	BE	BELIZE	BZ
BENIN	BJ	BERMUDA	BM
BHUTAN	BT	BOLIVIA	BO
BOTSWANA	BW	BOUVET ISLAND	BV
BRAZIL	BR	BRITISH INDIAN OCEAN TERRITORY	IO
BRUNEI DARUSSALAM	BN	BULGARIA	BG
BURKINA FASO	BF	BURMA	BU
BURUNDI	BI	BYELORUSSIAN SSR	BY
CAMEROON	CM	CANADA	CA
CAPE VERDE	CV	CAYMAN ISLANDS	KY

Country Name	Code	Country Name	Code
CENTRAL AFRICAN REPUBLIC	CF	CHAD	TD
CHILE	CL	CHINA	CN
CHRISTMAS ISLAND	CX	COCOS (KEELING) ISLANDS	CC
COLOMBIA	CO	COMOROS	KM
CONGO	CG	COOK ISLANDS	CK
COSTA RICA	CR	COTE D'IVOIRE	CI
CUBA	CU	CYPRUS	CY
CZECHOSLOVAKIA	CS	DENMARK	DK
DJIBOUTI	DJ	DOMINICA	DM
DOMINICAN REPUBLIC	DO	EAST TIMOR*	TP
ECUADOR	EC	EGYPT	EG
EL SALVADOR	SV	EQUATORIAL GUINEA	GQ
ETHIOPIA	ET	FALKLAND ISLANDS (MALVINAS)	FK
FAROE ISLANDS	FO	FIJI	FJ
FINLAND	FI	FRANCE	FR
FRENCH GUIANA	GF	FRENCH POLYNESIA	PF
FRENCH SOUTHERN TERRITORIES	TF	GABON	GA
GAMBIA	GM	GERMAN DEMOCRATIC REPUBLIC	DD
GERMANY, FEDERAL REPUBLIC OF	DE	GHANA	GH
GIBRALTAR	GI	GREECE	GR
GREENLAND	GL	GRENADA	GD
GUADELOUPE	GP	GUAM	GU
GUATEMALA	GT	GUINEA	GN
GUINEA-BISSAU	GW	GUYANA	GY
HAITI	HT	HEARD AND MC DONALD ISLANDS	HM
HONDURAS	HN	HONG KONG	HK
HUNGARY	HU	ICELAND	IS
INDIA	IN	INDONESIA	ID
IRAN (ISLAMIC REPUBLIC OF)	IR	IRAQ	IQ
IRELAND	IE	ISRAEL	IL
ITALY	IT	JAMAICA	JM
JAPAN	JP	JORDAN	JO
KAMPUCHEA, DEMOCRATIC	KH	KENYA	KE
KIRIBATI	KI	KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	KP
KOREA, REPUBLIC OF	KR	KUWAIT	KW
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LA	LEBANON	LB
LESOTHO	LS	LIBERIA	LR
LIBYAN ARAB JAMAHIRIYA	LY	LIECHTENSTEIN	LI
LUXEMBOURG	LU	MACAU	MO
MADAGASCAR	MG	MALAWI	MW
MALAYSIA	MY	MALDIVES	MV

Country Name	Code	Country Name	Code
MALI	ML	MALTA	MT
MARSHALL ISLANDS	MH	MARTINIQUE	MQ
MAURITANIA	MR	MAURITIUS	MU
MEXICO	MX	MICRONESIA	FM
MONACO	MC	MONGOLIA	MN
MONTSERRAT	MS	MOROCCO	MA
MOZAMBIQUE	MZ	NAMIBIA	NA
NAURU	NR	NEPAL	NP
NETHERLANDS	NL	NETHERLANDS ANTILLES	AN
NEUTRAL ZONE	NT	NEW CALEDONIA	NC
NEW ZEALAND	NZ	NICARAGUA	NI
NIGER	NE	NIGERIA	NG
NIUE	NU	NORFOLK ISLAND	NF
NORTHERN MARIANA ISLANDS	MP	NORWAY	NO
OMAN	OM	PAKISTAN	PK
PALAU	PW	PANAMA	PA
PAPUA NEW GUINEA	PG	PARAGUAY	PY
PERU	PE	PHILIPPINES	PH
PITCAIRN	PN	POLAND	PL
PORTUGAL	PT	PUERTO RICO	PR
QATAR	QA	REUNION	RE
ROMANIA	RO	RWANDA	RW
ST. HELENA	SH	SAINT KITTS AND NEVIS	KN
SAINT LUCIA	LC	ST. PIERRE AND MIQUELON	PM
SAINT VINCENT AND THE GRENADINES	VC	SAMOA	WS
SAN MARINO	SM	SAO TOME AND PRINCIPE	ST
SAUDI ARABIA	SA	SENEGAL	SN
SEYCHELLES	SC	SIERRA LEONE	SL
SINGAPORE	SG	SOLOMON ISLANDS	SB
SOMALIA	SO	SOUTH AFRICA	ZA
SPAIN	ES	SRI LANKA	LK
SUDAN	SD	SURINAME	SR
SVALBARD AND JAN MAYEN ISLANDS	SJ		
SWAZILAND	SZ	SWEDEN	SE
SWITZERLAND	CH	SYRIAN ARAB REPUBLIC	SY
TAIWAN, PROVINCE OF CHINA	TW	TANZANIA, UNITED REPUBLIC OF	TZ
THAILAND	TH	TOGO	TG
TOKELAU	TK	TONGA	TO
TRINIDAD AND TOBAGO	TT	TUNISIA	TN
TURKEY	TR	TURKS AND CAICOS ISLANDS	TC
TUVALU	TV	UGANDA	UG
UKRAINIAN SSR	UA	UNITED ARAB EMIRATES	AE
UNITED KINGDOM	GB	UNITED STATES	US

Country Name	Code	Country Name	Code
UNITED STATES MINOR OUTLYING ISLANDS	UM	URUGUAY	UY
USSR	SU	VANUATU	VU
VATICAN CITY STATE	VA	VENEZUELA	VE
VIET NAM	VN	VIRGIN ISLANDS (BRITISH)	VG
VIRGIN ISLANDS (U.S.)	VI	WALLIS AND FUTUNA ISLANDS	WF
WESTERN SAHARA*	EH	YEMEN	YE
YEMEN, DEMOCRATIC	YD	YUGOSLAVIA	YU
ZAIRE	ZR	ZAMBIA	ZM
ZIMBABWE	ZW		

* Provisional name.

A.3 T.61 Syntax

The set of T.61 characters supported are depicted in CCITT T.61. The XDS interface supports the full T.61 range as indicated there.

The optional use of additional character repertoires is permitted and provided for, according to the CCITT T.61 specification. However, neither the composition and allocation of these character sets nor the specific conformance requirements are specified here.

Some T.61 alphabetical characters have a two-byte representation. For example, a lower-case letter a with acute accent is represented by 0xc2 (code for acute accent) followed by 0x61 (code for lower-case a).

Only certain combinations of diacritical characters and basic letters are valid. They are shown in Table A-2.

Name	Repr.	Code	Valid Basic Letters Following
grave accent	`	0xc1	a, A, e, E, i, I, o, O, u, U
acute accent	´	0xc2	a,A,c,C,e,E,g,i,I,l,L,n,N,o,O,r,R, s,S,u,U,y,Y,z,Z
circumflex accent	ˆ	0xc3	a,A,c,C,e,E,g,G,h,H,i,I,j,J,o,O,s,S, u,U,w,W,y,Y
tilde	˜	0xc4	a,A,i,I,n,N,o,O,u,U
macron		0xc5	a,A,e,E,i,I,o,O,u,U
breve	˘	0xc6	a, A, g, G, u, U
dot above		0xc7	c,C,e,E,g,G,I,z,Z
umlaut	¨	0xc8	a,A,e,E,i,I,o,O,u,U,y,Y
ring	°	0xca	a,A,u,U
cedilla	¸	0xcb	c,C,G,k,K,l,L,n,N,r,R,s,S,t,T
double accent	"	0xcd	o, O, u, U
ogonek		0xce	a, A, e, E, i, I, u, U
caron	ˇ	0xcf	c,C,d,D,e,E,l,L,n,N,r,R,s,S,t,T,z,Z

Table A-2 Combinations of Diacritical Characters and Basic Letters

The non-spacing underline (code 0xcc) must be followed by a Latin alphabetical character; that is, a basic letter (a-z or A-Z), or a valid diacritical combination.

Bold characters cannot be reverse mapped.

All characters that stand alone must be followed by a space.

A.4 ISO 8859-1 (Latin-1) Syntax

GDS external interfaces support the use of ISO 8859-1 (Latin-1) syntax. When GDS receives 8859-1 input from administrative programs or XDS / XOM functions, it converts the input to T.61 format. Output is converted back to 8859-1 syntax.

Table A-3 shows the valid set of ISO 8859-1 characters. (The row headings indicate the lower four bits and the column headings show the higher four bits of the encoding in hexadecimal.)

	2	**3**	**4**	**5**	**6**	**7**	**A**	**B**	**C**	**D**	**E**	**F**
0	SP	0	@	P	‘	p	NBSP	°	À	Ð	à	ð
1	!	1	A	Q	a	q	¡	±	Á	Ñ	á	ñ
2	"	2	B	R	b	r	¢	^{2}	Â	Ò	â	ò
3	#	3	C	S	c	s	£	^{3}	Ã	Ó	ã	ó
4	$	4	D	T	d	t	¤	´	Ä	Ô	ä	ô
5	%	5	E	U	e	u	¥	µ	Å	Õ	å	õ
6	&	6	F	V	f	v	¦	¶	Æ	Ö	æ	ö
7	’	7	G	W	g	w	§	•	Ç	×	ç	÷
8	(	8	H	X	h	x	¨	¸	È	Ø	è	ø
9	)	9	I	Y	i	y	©	^{1}	É	Ù	é	ù
A	*	:	J	Z	j	z	ª	º	Ê	Ú	ê	ú
B	+	;	K	[	k	{	<<	>>	Ë	Û	ë	û
C	,	<	L	\	l	|	¬	$^1/_4$	Ì	Ü	ì	ü
D	-	=	M	]	m	}	SHY	$^1/_2$	Í	Ý	í	ý
E	.	>	N	ˆ	n	˜	®	$^3/_4$	Î	Þ	î	þ
F	/	?	O	_	o	DEL	¯	¿	Ï	ß	ï	ÿ

Table A-3 ISO 8859-1 (Latin-1) Code Set

A.4.1 Invalid Conversions from T.61 to ISO 8859-1 Syntax

If the following diacritical characters are followed by a space, they cannot be mapped from T.61 to the ISO 8859-1 codeset:

- c6 (breve)
- c7 (dot above)
- c8 (umlaut)
- ca (ring)
- cd (double accent)
- ce (ogonek)
- cf (caron)

Table A-4 on page 179 shows the invalid combinations of diacritical letters when converting from the T.61 codeset to the ISO 8859-1 codeset. GDS displays a ? (question mark) when it encounters a combination that is not defined in the ISO 8859-1 codeset.

Name	**Repr.**	**Code**	**Valid Basic Letters Following**
grave accent	`	0xc1	
acute accent	´	0xc2	c,C,g,l,L,n,N,r,R,s,S,z,Z
circumflex accent	ˆ	0xc3	c,C,g,G,h,H,j,J,s,S,w,W,y,Y
tilde	˜	0xc4	i,I,u,U
macron		0xc5	a,A,e,E,i,I,o,O,u,U
breve	˘	0xc6	a,A,g,G,u,U
dot above		0xc7	c,C,e,E,g,G,I,z,Z
umlaut	¨	0xc8	Y
ring	˚	0xca	u,U
cedilla	¸	0xcb	G,k,K,l,L,n,N,r,R,s,S,t,T
double accent	"	0xcd	o,O,u,U
ogonek		0xce	a,A,e,E,i,I,u,U
caron	ˇ	0xcf	c,C,d,D,e,E,l,L,n.N,r,R,s,S,t,T,z,Z

Table A-4 Invalid ISO 8859-1 Combinations of Diacritical Characters and Letters

A.4.2 Invalid Conversions from ISO 8859-1 to T.61 Syntax

The following characters cannot be converted from ISO 8859-1 syntax to T.61 syntax; GDS generates an error when it attempts to perform the conversion:

- 5c (\)
- 7b ({)
- 7d (})
- a6 (¦)
- a9 (©)
- ac (¬)
- ad (SHY)
- ae (®)
- b9 (^{1})

A.5 Metacharacters

Certain characters have special meaning to the directory services; these are known as *metacharacters*. Table A-5 lists and explains the CDS, GDS and DNS metacharacters.

Directory Service	Character	Meaning
CDS	/	Separates components of a name (atomic names).
	*	When used in the rightmost atomic name of a name, acts as a wildcard, matching zero or more characters.
	?	When used in the rightmost atomic name of a name, acts as a wildcard, matching exactly one character.
	\	Used where necessary in front of a / (slash), an * (asterisk) or a ? (question mark) to escape the character (indicates that the following character is not a metacharacter).
GDS	/	Separates Relative Distinguished Names (RDNs).
	,	Separates multiple attribute type/value pairs (attribute value assertions) within an RDN.
	=	Separates an attribute type and value in an attribute value assertion.
	\	Used in front of a / (slash), a , (comma) or an = (equal sign) to escape the character (indicates that the following character is not a metacharacter).
DNS	.	Separates elements of a name.

Table A-5 Metacharacters and Their Meaning

Some metacharacters are not permitted as normal characters within a name. For example, a \ (backslash) cannot be used as anything but an escape character in GDS.

A.6 Additional Rules

Table A-6 summarises major points about CDS, GDS and DNS character sets, metacharacters, restrictions, case-matching rules, internal storage of data, and ordering of elements in a name.

Table A-6 Summary of CDS, GDS and DNS Characteristics

Characteristic	CDS	GDS	DNS
Character Set	a to z, A to Z, 0 to 9 plus space and special characters shown in Figure A-1	a to z, A to Z, 0 to 9 plus . : ,	a to z, A to Z, 0 to 9 plus . -
Metacharacters	/ * ? \	/ , = \	.
Restrictions	Atomic names cannot contain a / (slash). The first atomic name following the global cell name (or /.: prefix) cannot contain an = (equal sign). For enumeration operations, a \ (backslash) must be used to escape any * (asterisk) or ? (question mark) character in the rightmost atomic name. Otherwise (if not escaped in the rightmost atomic name), the character is interpreted as a wildcard.	Relative distinguished names cannot begin or end with a / (slash). Attribute types must begin with an alphabetic character, can contain only alphanumerics, and cannot contain spaces. An alternative method of specifying attribute types is by object identifier, a sequence of digits separated by . (dots). A \ (backslash) must be used to escape a / (slash), a , (comma), and an = (equal sign) when using them as anything other than metacharacters. Multiple consecutive unescaped instances of / (slashes), , (commas), = (equal signs) and \ (back-slashes) are not allowed. Each attribute value assertion contains exactly one unescaped = (equal sign).	The first character must be alphabetic. The first and last characters cannot be a . (dot) or a - (dash). Cell names in DNS must contain at least one . (dot); they must be more than one level deep.

Characteristic	CDS	GDS	DNS
Case-Matching Rules	Case exact	Attribute types are matched case insensitive. The case-matching rule for an attribute value can be case exact or case insensitive, depending on the rule defined for its type at the DSA.	Case insensitive
Internal Representation	Case exact	Depends on the case-matching rule defined at DSA. If the rule says case insensitive, alphabetic characters are converted to all lower-case characters. Spaces are removed regardless of the case-matching rule.	Alphabetic characters are converted to all lower-case characters.
Ordering of Name Elements	Big endian (left to right from root to lower-level names).	Big endian (left to right from root to lower-level names).	Little endian (right to left from root to lower-level names).

A.7 Maximum Name and Attribute Sizes

Table A-7 lists maximum sizes for Directory Service names and attributes in CDS.

Notes:

1. Implementations must support these sizes and must not exceed these limits in remote operations.
2. The defined sizes determine the actual contents of the string, not including the terminating null.

Name Type	Maximum Size (octets)
CDS atomic name (character string between two slashes)	254
CDS full name (including global or local prefix, cell name and slashes separating atomic names)	1023
CDS attribute name	31
CDS class name	31
GDS distinguished name	1023
DNS relative name (character string between two dots)	64
DNS fully qualified name (sum of all relative names)	255

Table A-7 Maximum Sizes of Directory Service Names

Appendix B

Object Identifiers for CDS Attributes

The X / Open Directory Services (XDS) interface offers client application programmers the ability to create and maintain names in either CDS or GDS. Programmers also can create new CDS attribute names or GDS attribute type labels. In the DCE Directory Service, every CDS attribute name and GDS attribute type label has a corresponding unique number called an *object identifier*.

B.1 Origin of Object Identifiers

The purpose of object identifiers is to ensure uniqueness among the attribute types that many different applications generate and use. Object identifiers are typically obtained from a hierarchy of allocation authorities, the highest being the International Organization for Standardization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT). Individual application developers do not usually have to contact ISO or CCITT directly to obtain unique numbers. Application developers are more likely to request object identifiers from a person within their company who is in charge of allocating them. The company authority would in turn contact a higher authority to obtain a unique company prefix.

The hierarchy of allocation authorities is indicated by dots that separate portions of an object identifier. Each string of numbers delineated by dots represents a level of the allocation hierarchy, going left to right from the highest authority down. For example, the object identifier 1.3.22.1.1.2 consists of the following levels:

1	ISO
3	Identified organisation
22	Open Software Foundation
1	Distributed Computing Environment
1	Remote Procedure Call
2	RPC Object UUIDs

B.2 CDS Attributes Table

The following table lists the currently registered object identifiers for CDS attributes:

```
# The derivation of the stem of these object ids is:
#
#       {iso(1) identified-org(3) osf(22) dce(1) cds(3)}
#
#     OID       LABEL                   SYNTAX
#
1.3.22.1.3.10   CDS_Members             VT_GroupMember
1.3.22.1.3.11   CDS_GroupRevoke         VT_Timeout
1.3.22.1.3.12   CDS_CTS                 VT_Timestamp
1.3.22.1.3.13   CDS_UTS                 VT_Timestamp
1.3.22.1.3.15   CDS_Class               VT_byte (ASCII encoding, ISO 8859-1)
1.3.22.1.3.16   CDS_ClassVersion        VT_Version
1.3.22.1.3.17   CDS_ObjectUUID          VT_uuid
1.3.22.1.3.19   CDS_Replicas            VT_ReplicaPointer
1.3.22.1.3.20   CDS_AllUpTo             VT_Timestamp
1.3.22.1.3.21   CDS_Convergence         VT_small
1.3.22.1.3.22   CDS_InCHName            VT_small
1.3.22.1.3.23   CDS_ParentPointer       VT_ParentPointer
1.3.22.1.3.24   CDS_DirectoryVersion    VT_Version
1.3.22.1.3.25   CDS_UpgradeTo           VT_Version
1.3.22.1.3.27   CDS_LinkTarget          VT_FullName
1.3.22.1.3.28   CDS_LinkTimeout         VT_Timeout
1.3.22.1.3.30   CDS_Towers              VT_byte
1.3.22.1.3.32   CDS_CHName              VT_FullName
1.3.22.1.3.34   CDS_CHLastAddress       VT_byte
1.3.22.1.3.36   CDS_CHState             VT_small
1.3.22.1.3.37   CDS_CHDirectories       VT_CHDirectory
1.3.22.1.3.40   CDS_ReplicaState        VT_small
1.3.22.1.3.41   CDS_ReplicaType         VT_small
1.3.22.1.3.42   CDS_LastSkulk           VT_Timestamp
1.3.22.1.3.43   CDS_LastUpdate          VT_Timestamp
1.3.22.1.3.44   CDS_RingPointer         VT_uuid
1.3.22.1.3.45   CDS_Epoch               VT_uuid
1.3.22.1.3.46   CDS_ReplicaVersion      VT_Version
1.3.22.1.3.48   CDS_NSCellname          VT_char
1.3.22.1.3.52   CDS_GDAPointers         VT_gdaPointer
1.3.22.1.3.53   CDS_CellAliases         VT_GroupMember
1.3.22.1.3.54   CDS_ParentCellPointers  VT_ReplicaPointer
1.3.22.1.1.1    RPC_ClassVersion        VT_byte
1.3.22.1.1.2    RPC_ObjectUUIDs         VT_byte
1.3.22.1.1.3    RPC_Group               VT_byte (ASCII encoding, ISO 8859-1)
1.3.22.1.1.4    RPC_Profile             VT_byte (ASCII encoding, ISO 8859-1)
1.3.22.1.1.5    RPC_Codesets            VT_byte
1.3.22.1.5.1    SEC_RepUUID             VT_byte
```

The first column contains the object identifier (OID), the second column contains a label (the name to which the identifier is mapped), and the third column indicates the data type (defined in Chapter 11).

Appendix C

CDS Status and Error Codes

Table C-1 lists the set of defined status messages that may be returned by the CDS protocol.

Table C-1 CDS Status Messages

Status Name	Code	Description
[CDS_SUCCESS]	1	Operation completed successfully.
[CDS_INVALIDARGUMENT]	1001	Invalid argument.
[CDS_INVALIDNAME]	1002	Invalid name.
[CDS_NONSRESOURCES]	1003	Insufficient local resources to process request.
[CDS_NOCOMMUNICATION]	1004	Unable to communicate with any CDS server.
[CDS_ACCESSDENIED]	1005	Insufficient rights to perform requested operation.
[CDS_CANNOTAUTHENTICATE]	1006	Requesting principal could not be authenticated to the clearinghouse.
[CDS_CONFLICTINGARGUMENTS]	1007	Conflicting arguments specified.
[CDS_TIMEOUTNOTDONE]	1008	Timeout occurred, operation not performed.
[CDS_TIMEOUTMAYBEDONE]	1009	Timeout occurred, operation may have completed.
[CDS_ENTRYEXISTS]	1011	Specified fully-qualified global name already exists.
[CDS_UNKNOWNENTRY]	1012	Requested entry does not exist.
[CDS_NOTSUPPORTED]	1013	Requested function not supported by this version of the architecture.
[CDS_NOTIMPLEMENTED]	1014	Requested optional function is not implemented.
[CDS_INVALIDUPDATE]	1015	Specified attribute cannot be modified.
[CDS_UNKNOWNCLEARINGHOUSE]	1016	Specified clearinghouse does not exist.
[CDS_NOTAREPLICA]	1017	Specified clearinghouse does not contain a copy of the directory.
[CDS_ALREADYREPLICA]	1018	Specified clearinghouse already contains a copy of the directory.
[CDS_CRUCIALREPLICA]	1019	Cannot delete crucial replica.
[CDS_NOTEMPTY]	1020	Directory must be empty to be deleted.
[CDS_NOTLINKED]	1021	Specified name exists but is not a soft link.
[CDS_POSSIBLECYCLE]	1022	Possible cycle in soft links detected.
[CDS_DANGLINGLINK]	1023	Specified soft link points to non-existent entry.
[CDS_NOTAGROUP]	1024	Not a group.
[CDS_CLEARINGHOUSEDOWN]	1025	Requested clearinghouse exists but is not available.
[CDS_BADEPOCH]	1026	Directory replicas are not synchronised.
[CDS_BADCLOCK]	1027	Server clocks are not synchronised.
[CDS_DATACORRUPTION]	1028	Data corruption detected at clearinghouse.
[CDS_WRONGATTRIBUTETYPE]	1029	Specified attribute type is incorrect.
[CDS_MORETHANONEREPLICA]	1030	Replica set of specified directory contains more than one replica.
[CDS_CANTPUTHERE]	1031	Replica of specified directory cannot be created in old clearinghouse.

Status Name	Code	Description
[CDS_OLDSKULK]	1032	Skulk in progress terminated, superseded by more recent skulk.
[CDS_UNTRUSTEDCH]	1033	Server process has insufficient access to clearinghouse.
[CDS_VERSIONSKEW]	1034	Replica cannot be added to old clearinghouse.
[CDS_NEWVERSIONSKEW]	1035	Old replica cannot be included in new replica set.
[CDS_NOTNAMESERVER]	1100	Operation can only be performed on a server node.
[CDS_WRONGSTATE]	1101	Responding entity in wrong state to process requested operation.
[CDS_BADNICKNAME]	1102	Specified nickname already assigned to another namespace.
[CDS_LOCALONLY]	1103	This operation cannot be performed remotely.
[CDS_NOTROOT]	1104	This operation must be performed on master copy of root directory.
[CDS_NOTCHDIRECTORY]	1105	Specified directory does not allow clearinghouse name storage.
[CDS_ROOTLOST]	1106	Requested operation would result in lost connectivity to root directory.
[CDS_CANNOTUPGRADE]	1107	Cannot upgrade replica in old clearinghouse.
[CDS_UNDERSPECIFIEDNAME]	1108	Specified name is not stored in a clearinghouse.
[CDS_UNKNOWNATTRIBUTE]	1109	Requested attribute does not exist.
[CDS_NOTINCELL]	1110	The name provided is not a part of the current cell.
[CDS_NOT_AN_ALIAS]	1111	The name provided was not an alias or the current cell name.
[CDS_PREFERRED_EXISTS]	1112	You cannot supersede a primary cell alias with a normal cell alias of the same name.
[CDS_CANNOT_REM_PREFERRED]	1113	You cannot remove the current primary cell alias (current cell name).
[CDS_MISSING_ALLUPTO]	1114	Unable to modify the cell aliases because the root directory is missing the AllUpTo attribute.
[CDS_REM_NOT_SAFE]	1115	You cannot remove an alias until the primary alias is considered safe.
[CDS_MULT_PREFERRED]	1116	Multiple primary aliases have been detected in the CDS_CellAliases attribute.
[CDS_CANNOT_REM_CELLNAME]	1117	You cannot remove the current cellname alias.
[CDS_ALIASTOOLONG]	1118	The supplied cellname is required to be < 255 bytes in length.
[CDS_LOWDIRVERSION]	1119	For cellname commands, the CDS_DirectoryVersion attribute is required to be 4.0 or greater.
[CDS_ALIASCYCLE]	1120	The supplied cellname would result in a hierarchical cycle.
[CDS_MISSING_DIRECTORYVERSION]	1121	Unable to modify the cell aliases because the root directory is missing the DirectoryVersion attribute.
[CDS_ALIASTOOSHORT]	1122	The supplied cellname is required to have two simple names for an alias and three simple names for a child alias.
		The proposed parent name is not its cellname or primary alias.

[CDS_NOTPREFERREDHIERARCHY]	1123	

Status Name	Code	Description
[CDS_NOPREFERRED_EXISTS]	1124	There is no existing primary alias for the local cell.
[CDS_CLERKBUG]	1998	Implementation-specific error.
[CDS_NAMESERVERBUG]	1999	Software error detected in server.
[CDS_ACCESSVIOLATION]	2000	Access violation.
[CDS_RESOURCEERROR]	2001	Insufficient resources to process request.
[CDS_NOROOM]	2025	Insufficient room in buffer.

Appendix D

CDS IDL Definitions

This appendix gives the IDL specification of the CDS remote interface, including:

cds_clerkserver.idl	The remote interface for the transaction protocol.
cds_solicit.idl	The remote interface for the solicitation protocol.
cds_types.idl	Header file for basic CDS types.
id_base.idl	Base type definitions for identities.

Notes:

1. The listings of *.idl definitions in this appendix provide supplementary information. In particular, the ordering of operations is significant for conforming implementations. For further information about the semantics of operations and encodings refer to Chapter 11 and Chapter 12.
2. The **cds_clerkserver.idl** definition contains the following operations that are not further specified in this document:

   cds_AddReplica
   cds_AllowClearinghouses
   cds_Combine
   cds_DisallowClearinghouses
   cds_DoUpdate
   cds_LinkReplica
   cds_ModifyReplica
   cds_NewEpoch
   cds_RemoveReplica
   cds_Skulk
   cds_TestGroup

   These operations are used by the CDS replication service that is not specified in this document. However, in order to preserve the ordering of IDL operations, it is necessary to maintain these operations in the *.idl definition.

D.1 cds_clerkserver.idl

```
[uuid(257df1c9-c6d3-11ca-8554-08002b1c8f1f), version(1.0)]

interface cds_clerkserver

{
import "dce/cds_types.idl";
import "dce/id_base.idl";

error_status_t cds_AddReplica(
    [in] handle_t h,
    [in] cds_FullName_t  *directory_p,
    [in] unsigned small type,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_AllowClearinghouses(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
    );

error_status_t cds_Combine(
    [in] handle_t h,
    [in] uuid_t *dir_p,
    [in] cds_Timestamp_t *skulk_p,
    [in] cds_Timestamp_t *allupto_p,
    [in] uuid_t *epoch_p,
    [in] cds_FullName_t *to_p,
    [out] uuid_t *next_p,
    [out] cds_status_t *cds_status_p
    );

error_status_t cds_CreateChild(
    [in] handle_t h,
    [in,out] cds_Progress_t *Progress_p,
    [in,ptr]sec_id_foreign_t *user_p,
    [in] uuid_t *childID_p,
    [in] cds_Set_t *replicaset_p,
    [out] uuid_t *parentID_p,
    [out] cds_status_t *cds_status_p
   );

error_status_t cds_CreateDirectory(
    [in] handle_t  h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_Timestamp_t  *actual_ts_p,
    [out] cds_status_t *cds_status_p
    );
```

```
error_status_t cds_CreateSoftLink(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] cds_FullName_t  *target_p,
    [in,ptr] cds_Timeout_t  *linkTimeout_p,
    [out] cds_Timestamp_t  *actual_ts_p,
    [out] cds_status_t *cds_status_p
    );

error_status_t cds_CreateObject(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in,ptr] cds_Name_t  *class_p,
    [in,ptr] cds_Version_t  *version_p,
    [in, ptr] uuid_t        *uuid_p,
    [out] cds_Timestamp_t *acutal_ts_p,
    [out] cds_status_t *cds_status_p
    );

error_status_t cds_DeleteChild(
    [in] handle_t h,
    [in,out] cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
   );

error_status_t cds_DeleteObject(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
    );

error_status_t cds_DeleteSoftLink(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
    );

error_status_t cds_DeleteDirectory(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_DisallowClearinghouses(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_DoUpdate(
    [in] handle_t h,
    [in] uuid_t *epoch_p,
    [in] cds_UpdatePkt_t *update_p,
    [out] cds_status_t *cds_status_p
    );
```

```
[idempotent] error_status_t cds_EnumerateAttributes(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] unsigned small type,
    [in] cds_Name_t  *context_p,
    [in] unsigned32 max_size,
    [in,out,ptr] cds_SetP_t *attr_set,
    [out] unsigned small *wholeset_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_EnumerateChildren(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] cds_Name_t  *wild_p,
    [in] cds_Name_t  *context_p,
    [in] unsigned32 max_size,
    [in,out,ptr] cds_SetP_t *name_set,
    [out] unsigned small *wholeset_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_EnumerateObjects(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] cds_Name_t  *wild_p,
    [in] cds_Name_t  *context_p,
    [in] cds_Name_t  *class_p,
    [in] unsigned32 max_size,
    [in,out,ptr] cds_SetP_t *name_set,
    [out] unsigned small *wholeset_p,
    [in,out] unsigned small *returnClass_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_EnumerateSoftLinks(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] cds_Name_t  *wild_p,
    [in] cds_Name_t  *context_p,
    [in] unsigned32 max_size,
    [in,out,ptr] cds_SetP_t *name_set,
    [out] unsigned small *wholeset_p,
    [out] cds_status_t *cds_status_p
    );
```

```
[idempotent] error_status_t cds_LinkReplica(
    [in] handle_t h,
    [in,out] cds_Progress_t *Progress_p,
    [in] uuid_t *directory_p,
    [in] cds_Update_t *update_p,
    [out] uuid_t *epoch_p,
    [out] uuid_t *ring_p,
    [out] cds_Version_t *rpversion_p,
    [out] cds_status_t *cds_status_p
   );

[idempotent] error_status_t cds_ModifyAttribute(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] unsigned small type,
    [in] cds_Update_t  *update_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_ModifyReplica(
    [in] handle_t h,
    [in] uuid_t *dir_p,
    [in] cds_Update_t *update_p,
    [out] cds_status_t *cds_status_p
   );

error_status_t cds_NewEpoch(
    [in] handle_t h,
    [in] cds_FullName_t *dirname_p,
    [in] cds_Set_t *readonlies_p,
    [in] cds_Set_t *secondaries_p,
    [out] cds_status_t *cds_status_p
    );

const unsigned small RA_none = 1;
const unsigned small RA_single = 2;
const unsigned small RA_set = 3;
const unsigned small RA_wholeSet = 4;

typedef struct {
    cds_Name_t name;
    union switch (unsigned small type) {
    case RA_none:
        ;
    case RA_single:
        [ptr] cds_SetMember_t *single_p;
    case RA_set:
        [ptr] cds_Set_t *set_p;
    } value;
} cds_WE_entry_t;
```

```
typedef struct {
    unsigned small numberOfAttributes;
    [size_is(numberOfAttributes)] cds_WE_entry_t entry[];
} cds_WholeEntry_t;

typedef union switch (unsigned small returningToClerk) {
    case RA_none:
        ;
    case RA_single:
        [ptr] cds_SetMember_t *value_single_p;
    case RA_set:
        [ptr] cds_Set_t *value_set_p;
    case RA_wholeSet:
        [ptr] cds_WholeEntry_t *wholeEntry_p;
} cds_RA_value_t;

[idempotent] error_status_t cds_ReadAttribute(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] unsigned small type,
    [in] cds_Name_t  *att_p,
    [in] cds_Timestamp_t  *context_p,
    [in] unsigned32 max_size,
    [in] unsigned32 maybemore,
    [out] cds_RA_value_t *value_p,
    [out] unsigned small *wholeset_p,
    [out] cds_status_t *cds_status_p
    );

error_status_t cds_RemoveReplica(
    [in] handle_t  h,
    [in] cds_FullName_t  *directory_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_ResolveName(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_Skulk(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [out] cds_status_t *cds_status_p
    );
```

```
[idempotent] error_status_t cds_TestAttribute(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] unsigned small type,
    [in] cds_Name_t  *att_p,
    [in] cds_AtomicValue_t  *value_p,
    [out] unsigned small  *result_p,
    [out] cds_status_t *cds_status_p
    );

[idempotent] error_status_t cds_TestGroup(
    [in] handle_t h,
    [in,out]cds_Progress_t *Progress_p,
    [in] cds_FullName_t  *member_p,
    [in,out] uuid_t *loop_p,
    [in,out] unsigned small *direct_p,
    [out] unsigned small *result_p,
    [in,out,ptr] cds_TimeoutP_t *outTimeout_p,
    [out] cds_status_t *cds_status_p
    );

}
```

D.2 cds_solicit.idl

```
[uuid(d5579459-8bca-11ca-b771-08002b1c8f1f), version(1.0)]
interface cds_solicit
{
import "cds_types.idl";

typedef [ref]cds_ReplicaPointer_t *cds_ReplicaPointerP_t;

typedef struct {
    unsigned short ch_length;
    [size_is(ch_length)] cds_ReplicaPointerP_t ch_members[];
} cds_CH_t;

typedef [ptr]cds_CH_t *cds_CHP_t;

[broadcast,maybe] void cds_Solicit(
    [in] handle_t h);

[broadcast,maybe] void cds_Advertise(
    [in] handle_t h,
    [in] cds_FullName_t *cellname_p,
    [in] uuid_t  cell_diruid,
    [in] cds_CH_t *nscle_p);

[idempotent] error_status_t cds_SolicitServer(
    [in] handle_t h,
    [out] cds_FullName_t *cellname_p,
    [out] uuid_t  *cell_diruid_p,
    [in,out,ptr] cds_CHP_t *nscle_p);

}
```

D.3 cds_types.idl

```
interface cds_types
{

#ifdef DCE_SEC
import "dce/aclbase.idl";
#endif

const small VT_none = 0;
const small VT_long = 1;
const small VT_short = 2;
const small VT_small = 3;
const small VT_uuid = 4;
const small VT_Timestamp = 5;
const small VT_Timeout = 6;
const small VT_Version = 7;
const small VT_char = 8;
const small VT_byte  = 9;
const small VT_ReplicaPointer = 10;
const small VT_GroupMember = 11;
const small VT_ParentPointer = 12;
const small VT_FullName = 13;
const small VT_CHDirectory = 14;
const small VT_ASN1 = 15;
const small VT_DACL = 16;
const small VT_gdaPointer = 18;

typedef small ValueType_t;

/*
 * CDS timestamp - 6 byte node id followed by time based on smithsonian
 * zero
 */
typedef struct {
        byte               ts_node[6];
        unsigned hyper int ts_time;
} cds_Timestamp_t;

/*
 * 2 utc values.  First is absolute time, second is relative
 */
typedef struct  {
        byte    to_expire[16];
        byte    to_extend[16];
} cds_Timeout_t;

typedef [ptr]cds_Timeout_t *cds_TimeoutP_t;
```

```
/*
 * Can be used for software version numbers
 */
typedef struct {
        unsigned small int ver_major;
        unsigned small int ver_minor;
} cds_Version_t;

/*
 * Represents Simplenames, attributenames and class values as input to
 * RPC routines.  These names are passed in their internal opq format.
 * within the nm_name array.
 */
typedef struct {
        unsigned short int          nm_length;
        [length_is(nm_length)] byte nm_name[257];
} cds_Name_t;

/*
 * The global root and the string representation of the pathname.
 * CDS server-server operations may use a directory uuid in the fn_root
 * and the fn_name is the entry name relative to fn_root (ie, not global)
 * fn_name does not include terminating null
 */
typedef struct {
        uuid_t          fn_root;
        long int        fn_length;
        [length_is(fn_length)] char fn_name[1023];
} cds_FullName_t;

/*
 * Error status return.  On Unknownentry, the er_name may be filled in
 * with the last name the server successfully accessed.
 */
typedef struct {
        unsigned long int   er_status;
        [ptr]cds_FullName_t *er_name;
} cds_status_t;

/*
 * Internal CDS structure that describes the replicas of directory
 *
 * const small RT_master = 1;
 * const small RT_readOnly = 3;
 * const small RT_gda = 4;
 */
typedef struct {
   unsigned small int  rp_type;
   uuid_t              rp_chid;
   cds_FullName_t      rp_chname;
   unsigned long int   rp_length;
   [size_is(rp_length)] byte rp_towers[];
} cds_ReplicaPointer_t;
```

```
/*
 * Internal CDS structure to describe the members of a CDS group
 * Members may by other group names in which isaGroup would be true
 */
typedef struct {
  boolean          gm_isaGroup;
  cds_FullName_t   gm_member;
} cds_GroupMember_t;

/*
 * Internal CDS structure that describes the parent of a directory
 * The timeout value is used to update replica pointers in the
 * child ptr.
 */
typedef struct {
  uuid_t           pp_parentID;
  cds_Timeout_t    pp_timeout;
  cds_FullName_t   pp_myName;
} cds_ParentPointer_t;

/*
 * Uppointers to the GDA
 */
typedef struct {
  cds_Timeout_t        gp_timeout;
  cds_ReplicaPointer_t gp_replica;
} cds_gdaPointer_t;

/*
 * Internal CDS attribute that defines the directories within a
 * clearinghouse
 */
typedef struct {
   uuid_t          cp_dirID;
   cds_FullName_t  cp_directory;
} cds_CHDirectory_t;

/*
 * Structure for transporting opaque variable length user data
 */
typedef struct {
   unsigned short int        op_length;
   [size_is(op_length)] byte op_array[];
} cds_OpenByte_t;

/*
 * Structure for transporting char variable length user data
 */
typedef struct {
   unsigned short int        op_length;
   [size_is(op_length)] char op_array[];
} cds_OpenChar_t;
```

```
/* All the data types CDS recognizes */
typedef union switch (ValueType_t av_valuetype) av_val
        {
        case VT_none:           ;
        case VT_long:           long int  av_long;
        case VT_short:          short int av_short;
        case VT_small:          small int av_small;
        case VT_uuid:           uuid_t    av_uuid;
        case VT_Timestamp:      cds_Timestamp_t av_timestamp;
        case VT_Timeout:        cds_Timeout_t av_timeout;
        case VT_Version:        cds_Version_t av_version;
        case VT_char:           [ptr]cds_OpenChar_t *av_char_p;
        case VT_ASN1:
        case VT_byte:           [ptr]cds_OpenByte_t *av_byte_p;
/* The remaining types are for internal CDS use only */
        case VT_ReplicaPointer: [ptr]cds_ReplicaPointer_t *av_rp_p;
        case VT_GroupMember:    [ptr]cds_GroupMember_t *av_gm_p;
        case VT_ParentPointer:  [ptr]cds_ParentPointer_t *av_pp_p;
        case VT_FullName:       [ptr]cds_FullName_t *av_fullname_p;
        case VT_CHDirectory:    [ptr]cds_CHDirectory_t *av_cp_p;
#ifdef DCE_SEC
        case VT_DACL:           [ptr]sec_acl_t         *av_acl_p;
#endif
        case VT_gdaPointer:     [ptr]cds_gdaPointer_t  *av_gda_p;
} cds_AtomicValue_t;

/*
 * Set member contains a present/absent flag, timestamp member
 * was created and the value
 * The flag is actually a bitmask with only bit 0 currently in use.
 * const small SM_present=1;
 * const small SM_absent=0;
 */
typedef struct {
    unsigned small int sm_flag;
    cds_Timestamp_t    sm_ts;
    cds_AtomicValue_t  sm_value;
} cds_SetMember_t;

/*
 * Sets describe single or set valued attributes. They contain a
 * a list of members
 * const small AT_none=1;
 * const small AT_single=2;
 * const small AT_set=3;
 */
typedef struct {
    unsigned small int set_type;
    unsigned short int set_length;
    [size_is(set_length)] cds_SetMember_t set_members[];
} cds_Set_t;
typedef [ptr]cds_Set_t *cds_SetP_t;
```

```
/*
 * Progress record is used to direct the clerk from one server
 * to another.
 * Flags is a bitmask.
 * const small PR_done=1;   Got results
 * const small PR_up =2;    returning ptr up the tree
 * const small PR_linked=4;  Link was found in name
 * const small PR_hitLink=8;    Link just found, new name returned
 * const small PR_ignoreState=16; ignore directory state
 * const small PR_directory =32;
 */
typedef struct {
    unsigned small int  pr_flags;
    [ptr]cds_Timeout_t  *pr_timeout;
    cds_FullName_t      pr_unresolved;
    cds_FullName_t      pr_resolved;
    [ptr]cds_Set_t      *pr_replicas_p;
} cds_Progress_t;

/*
 * Update an attribute
 * const small UD_present=1;
 * const small UD_absent=2;
 */
typedef struct {
    small int           ud_operation;
    cds_Timestamp_t     ud_timestamp;
    unsigned small int  ud_type;        /* attribute type */
    byte                ud_attribute[33];
    cds_AtomicValue_t   ud_value;
} cds_Update_t;

/*
 * Structure used to bundle multiple updates to one entry.  Used in
 * DoUpdate function.  Name is relative to a directory.
 * const small int ET_directory=1;
 * const small int ET_object=2;
 * const small int ET_childPointer=3;
 * const small int ET_softlink=4;
 * const small int ET_clearinghouse=5;
 * const small int ET_anyDirectName=6;
 * const small int ET_firstLink=7;
 * const small int ET_dirOrObj=8;
 */
typedef struct {
    cds_FullName_t      pkt_name;
    cds_Timestamp_t     pkt_cts;
    small int           pkt_type;       /* EntryType */
    unsigned small int  pkt_cnt;
    [size_is(pkt_cnt)] cds_Update_t pkt_data[];
} cds_UpdatePkt_t;

}
```

D.4 id_base.idl

```
/* Identity base type definitions
**
*/
[
    uuid(47EAABA3-3000-0000-0D00-01DC6C000000)
]
interface sec_id_base {
    import "dce/nbase.idl";
    /*  s e c _ i d _ t
     *
     *  A sec_id_t is the basic unit for identifying principals or groups etc.
     *  It contains the uuid (object handle for the identity) and an optional
     *  printstring name.
     *
     *  This datatype requires a destructor function since the printstring
     *  name is dynamically allocated.  Generally this datatype is embedded
     *  in other datatypes (like acl's) which have their own destructor which
     *  will release the printstring storage as well.
     */
    typedef struct {
        uuid_t              uuid;
        [ string,ptr ] char *name;
    } sec_id_t;
    /* s e c _ i d _ f o r e i g n _ t
     *
     *  A foreign id (sec_id_foreign_t) is an identity from a foreign realm.
     *  It contains a sec_id_t for the foreign user or group and a sec_id_t
     *  for the foreign realm.
     */
    typedef struct {
        sec_id_t            id;
        sec_id_t            realm;
    } sec_id_foreign_t;
    /* Set of groups all associated with the same foreign cell */
    typedef struct {
        sec_id_t realm;
        unsigned16 num_groups;
        [size_is(num_groups), ptr]
            sec_id_t *groups;
    } sec_id_foreign_groupset_t;
    /* s e c _ i d _ p a c _ f o r m a t _ t
     *
     *  A format label to indicate which EPAC format is being used.
     */
    typedef enum {
        sec_id_pac_format_v1
    } sec_id_pac_format_t;
    /* s e c _ i d _ p a c _ t      (sec_id_pac_format_v1)
     *
     *  A privilege attribute certificate contains the principal’s identity
     *  along with the current group and concurrent group set.  The pac is
     *  generally not directly visible to applications.  It is normally included
     *  in an authentication ticket where it appears as a sealed certificate.
     *  Anticipating future systems that wish to extend the EPAC structure, we
     *  include a format label (which takes the value sec_id_pac_format_t)
     *  and we also add version specific encodings of the type.  Applications
     *  that wish to provide persistent storage of a EPAC should
     *  use the version specific type name, applications that wish the latest
     *  version should use the simple sec_id_pac_t generic type name.
     *
     *  The pac contains an "authenticated" field which when set indicates that
     *  the certificate was obtained from an authenticated source.  When unset
     *  the certificate should not be trusted.  (The field is unset - false -
     *  when it is obtained from the rpc_auth layer and the "assert_id" mode
     *  was selected.  This indicates that no authentication protocol was
     *  actually used in the rpc, the identity was simply transmitted from the
     *  caller to the callee.  If an authentication protocol was used, then
     *  the flag is set to true)
     */
    typedef struct {
        sec_id_pac_format_t  pac_type;
        boolean32            authenticated;
        sec_id_t             realm;
        sec_id_t             principal;
        sec_id_t             group;
        unsigned16           num_groups;
        unsigned16           num_foreign_groups;
        [size_is(num_groups), ptr]
            sec_id_t         *groups;
        [size_is(num_foreign_groups), ptr]
            sec_id_foreign_t *foreign_groups;
    } sec_id_pac_t, sec_id_pac_format_v1_t;

    /* s e c _ i d _ p i c k l e d _ p a c _ t
     *
     *  An pickled privilege attribute certificate is a labeled pac.  It
     *  contains an ndr_format_t label which describes the pickle format.
     *  Generally a pickled pac occurs in an authentication ticket where
     *  the data is also encrypted.
     */
    typedef struct {
        ndr_format_t         format_label;
        unsigned32           num_bytes;
        [size_is(num_bytes)]
            byte             pickled_data[];
    } sec_id_pickled_pac_t;

}
```

GDS Structure Rule Table

The Structure Rule Table (SRT) supplied with the GDS default schema contains the following entries. This SRT is defined for a DSA.

Rule Number	Superior Rule Number	Acronyms of Naming Attribute	Acronym of Structural Object Class
1	0	CN	SCH
2	0	C	C
3	2	O	ORG
4	3	OU	OU
5	4	CN	ORP
6	4	CN	ORR
7	4	CN	APP
8	4	CN	MDL
9	4	CN,OU	ORP
10	7	CN	APE
11	7	CN	DSA
12	7	CN	MMS
13	7	CN	MTA
14	7	CN	MUA
15	7	CN	DNA
16	2	L	LOC
17	16	CN	REP
18	16	CN,STA	REP

Table E-1 SRT Entries for a DSA

For an explanation of the acronyms of the naming attribute, see Table G-1 on page 209. For an explanation of the acronyms of object classes, see Table F-1 on page 207.

The following SRT is defined for the GDS administration programs.

Rule Number	Superior Rule Number	Acronyms of Naming Attribute	Acronym of Structural Object Class
1	0	CN	SCH
2	0	C	C
3	2	O	ORG
4	3	OU	OU
5	4	CN	ORP,APP,ORR,MDL
6	4	CN, OU	ORP
7	5	CN	DSA,APE,MMS,MTA,MUA,DNA
8	2	L	LOC
9	8	CN	REP
10	8	CN,STA	REP

Table E-2 SRT Entries for GDS

Appendix F

GDS Object Class Table

The object class table (OCT) supplied with the GDS default schema contains the following entries:

Table F-1 OCT Entries

Acr. of Object Class	Acr. of Super Classes	Object ID	Name of Object Class	Kind of Object Class	File No.	Acronym of Auxiliary Object Classes	Acronym of Mandatory Attributes	Acronym of Optional Attributes
TOP		85.6.0	Top	Abstract	-1	-	OCL	
ALI	TOP	85.6.1	Alias	Alias	-1	-	AON	
C	GTP	85.6.2	Country	Structural	1	-	C	DSC SG CDC CDR
LOC	GTP	85.6.3	Locality	Structural	4	-		DSC L SPN STA SEA SG CDC CDR
ORG	GTP	85.6.4	Organization	Structural	1	-	O	DSC L SPN STA PDO PA PC POB FTN IIN TN TTI TXN X1A PDM DI RA SEA UP BC SG CDC CDR
OU	GTP	85.6.5	Organizational-Unit	Structural	1	-	OU	DSC L SPN STA PDO PA PC POB FTN IIN TN TTI TXN X1A PDM DI RA SEA UP BC SG CDC CDR
PER	GTP	85.6.6	Person	Abstract	-1	-	CN SN	DSC TN SEA UP CDC CDR
ORP	PER	85.6.7	Organizational-Person	Structural	1	CA MUS SAU		L SPN STA PDO PA PC POB FTN IIN TTI TXN X1A PDM DI RA OU TIT
ORR	GTP	85.6.8	Organizational-Role	Structural	1	-	CN	DSC L SPN STA PDO PA PC POB FTN IIN TN TTI TXN X1A PDM DI RA OU SEA RO CDC CDR
GON	GTP	85.6.9	Group-of-Names	Structural	3	-	CN MEM	DSC O OU SEA BC OWN CDC CDR
REP	PER	85.6.10	Residential-Person	Structural	4	-	L	SPN STA PDO PA PC POB FTN IIN TTI TXN X1A PDM DI RA BC

Acr. of Object Class	Acr. of Super Classes	Object ID	Name of Object Class	Kind of Object Class	File No.	Acronym of Auxiliary Object Classes	Acronym of Mandatory Attributes	Acronym of Optional Attributes
APP	GTP	85.6.11	Application-Process	Structural	2	-	CN	DSC L OU SEA CDC CDR
APE	GTP	85.6.12	Application-Entity	Structural	2	-	CN PSA	DSC L O OU SEA SAC CDC CDR
DSA	APE	85.6.13	Directory-Service-Agent	Structural	2	-		AM KNI PN UP
DEV	GTP	85.6.14	Device	Structural	2	-	CN	DSC L O OU SEA OWN SER CDC CDR
SAU	TOP	85.6.15	Strong-Auth.-User	Auxiliary	-1	-	UC	
CA	TOP	85.6.16	Certification-Authority	Auxiliary	-1	-	CAC CRL ARL	CCP
MDL	GTP	86.5.1.0	MHS-Distribution-List	Structural	2	-	CN MDS MOA	DSC O OU SEA OWN MDT MDE MDM MPD
MMS	APE	86.5.1.1	MHS-Message-Store	Structural	2	-		OWN MSO MSA MSC
MTA	APE	86.5.1.2	MHS-Mess-Transfer-Agent	Structural	2	-		OWN MDL
MUS	TOP	86.5.1.3	MHS-User	Auxiliary	-1	-	MOA	MDL MDT MDE MMS MPD
MUA	APE	86.5.1.4	MHS-User-Agent	Structural	2	-		OWN MDL MDT MDE MOA
SCH	GTP	43.12.2. 1107.1.3. 6.0	Schema	Structural	0	-	CN	TST SRT OCT AT
GTP	TOP	None	GDS-Top	Abstract	-1	-		ACL MK

For an explanation of acronyms of the mandatory attribute and the optional attribute, see Table G-1 on page 209.

Notes:

- Although the object class **Locality** (acronym = LOC) does not have a specific set of mandatory attributes, either the **Locality-Name** (acronym = L) or the **State-or-Province-Name** (acronym = SPN) attribute must be present (for further information, see Appendix G.
- Every object class inherits all mandatory and optional attributes of its superior object classes.
- The object classes **Group-of-Names** (GON) and **Device** (DEV) are object classes defined in the X.500 standard. They are part of the OCT, but they are not assigned to any structure rule of the SRT of the GDS default schema.

Appendix G

GDS Attribute Table

The attribute table (AT) supplied with the GDS default schema contains the following entries (see the tables at the end of this appendix for explanations of abbreviations and values):

Table G-1 AT Entries

Acr. of Attr.	Object ID	Name of Attribute	Lower Bound	Upper Bound	Max. No. of Values	Syntax	Phon. Flag	Access Class	Index Level
OCL	85.4.0	Object-Class	1	28	0	2	0	0	0
AON	85.4.1	Aliased-Object-Name	1	1024	1	1	0	0	0
KNI	85.4.2	Knowledge-Information	1	1024	0	4	0	0	0
CN	85.4.3	Common-Name	1	64	2	4	1	0	1
SN	85.4.4	Surname	1	64	2	4	1	0	0
SER	85.4.5	Serial-Number	1	64	2	5	0	0	0
C	85.4.6	Country-Name	2	2	1	1010	1	0	1
L	85.4.7	Locality-Name	1	128	2	4	1	0	1
SPN	85.4.8	State-or-Province-Name	1	128	2	4	1	0	0
STA	85.4.9	Street-Address	1	128	2	4	1	0	0
O	85.4.10	Organization-Name	1	64	2	4	1	0	1
OU	85.4.11	Org.-Unit-Name	1	64	2	4	1	0	1
TIT	85.4.12	Title	1	64	2	4	1	0	0
DSC	85.4.13	Description	1	1024	0	4	0	0	0
SG	85.4.14	Search-Guide	1	256	0	1000	0	0	0
BC	85.4.15	Business-Category	1	128	2	4	1	0	0
PA	85.4.16	Postal-Address	1	180	2	1001	1	1	0
PC	85.4.17	Postal-Code	1	40	2	4	1	0	0
POB	85.4.18	Post-Office-Box	1	40	2	4	1	0	0
PDO	85.4.19	Phys.-Deliv.-Office-Name	1	128	2	4	1	0	0
TN	85.4.20	Telephone-Number	1	32	0	12	0	0	0
TXN	85.4.21	Telex-Number	1	26	0	1002	0	0	0
TTI	85.4.22	TTX-Terminal-Identifier	1	1024	0	1003	0	0	0
FTN	85.4.23	Fax-Telephone-Number	1	37	0	1004	0	0	0
X1A	85.4.24	X121-Address	1	15	0	6	0	0	0
IIN	85.4.25	Internat.-ISDN-Number	1	16	0	6	0	0	0
RA	85.4.26	Registered-Address	1	180	2	1001	1	0	0
DI	85.4.27	Destination-Indicator	1	128	2	4	1	0	0

Acr. of Attr.	Object ID	Name of Attribute	Lower Bound	Upper Bound	Max. No. of Values	Syntax	Phon. Flag	Access Class	Index Level
PDM	85.4.28	Preferred-Delivery-Method	1	10	1	1005	0	0	0
PSA	85.4.29	Presentation-Address	1	268	1	1006	0	0	0
SAC	85.4.30	Suppl.-Applic.-Context	1	28	2	2	0	0	0
MEM	85.4.31	Member	1	1024	0	1	0	0	0
OWN	85.4.32	Owner	1	1024	0	1	0	0	0
RO	85.4.33	Role-Occupant	1	1024	0	1	0	0	0
SEA	85.4.34	See-Also	1	1024	0	1	0	0	0
UP	85.4.35	User-Password	0	128	2	1011	0	2	0
US	85.4.36	User-Certificate	1	3024	0		0	0	0
CAC	85.4.37	CA-Certificate	1	3024	0		0	0	0
ARL	85.4.38	Authority-Revocation-List	1	32503	0		0	0	0
CRL	85.4.39	Certificate.-Revoc.-List	1	32503	0		0	0	0
CCP	85.4.40	Cross-Certificate-Pair	1	6056	0		0	0	0
MDL	86.5.2.0	MHS-Deliv.-Cont.-Length	4	4	1	9	0	0	0
MDT	86.5.2.1	MHS-Deliv.-Cont.-Types	1	28	4	2	0	0	0
MDE	86.5.2.2	MHS-Deliverable-EITs	1	28	8	2	0	0	0
MDM	86.5.2.3	MHS-DL-Members	1	3596	0	102	0	0	0
MDS	86.5.2.4	MHS-DL-Submit-Permission	1	3604	0	100	0	0	0
MMS	86.5.2.5	MHS-Message-Store	1	1024	1	1	0	0	0
MOA	86.5.2.6	MHS-OR-Address	1	2564	0	101	0	0	0
MPD	86.5.2.7	MHS-Pref.-Deliv.-Meth.	1	10	1	103	0	0	0
MSA	86.5.2.8	MHS-Supp.-Autom.-Action	1	28	4	2	0	0	0
MSC	86.5.2.9	MHS-Supp.-Content-Types	1	28	4	2	0	0	0
MSO	86.5.2.10	MHS-Supp.-Optional-Attr.	1	28	0	2	0	0	0
MK	43.12.2.11 07.1.3.4.0	Master-Knowledge	1	1024	1	1	0	0	0
ACL	43.12.2.11 07.1.3.4.1	Access-Control-List	1	20500	1	10000	0	0	0
TST	43.12.2.11 07.1.3.4.2	Time-Stamp	11	18	1	11	0	0	0
SRT	43.12.2.11 07.1.3.4.4	Structure-Rule-Table	1	29	0	5	0	0	0
OCT	43.12.2.11 07.1.3.4.5	Object-Class-Table	1	397	0	5	0	0	0

Acr. of Attr.	Object ID	Name of Attribute	Lower Bound	Upper Bound	Max. No. of Values	Syntax	Phon. Flag	Access Class	Index Level
AT	43.12.2.11 07.1.3.4.6	Attribute-Table	1	101	0	5	0	0	0
CDC	43.12.2.11 07.1.3.4.13	CDS-Cell	1	284	1	10	0	0	0
CDR	43.12.2.11 07.1.3.4.14	CDS-Replica	1	905	0	10	0	0	0
PN	43.12.2.11 07.1.3.4.15	Principal-Name	1	1024	1	5	0	0	0
AM	43.12.2.11 07.1.3.4.16	Authentication-Mechanism	4	4	7	9	0	0	0
AA	43.22.2.1.2.1.1	Alternate-Address	1	800	0	10	0	0	0

Explanations

The following tables provide explanations of the syntaxes, phonetic flags, maximum number of values, and access classes used in Table G-1 on page 209.

Digit	Explanation
0	Any Syntax
1	Distinguished Name
2	Object Identifier Syntax
3	Case Exact String
4	Case Ignore String
5	Printable String
6	Numeric String
7	Case Ignore List
8	Boolean Syntax
9	Integer Syntax
10	Octet String
11	UTC Time
12	Telephone Number Syntax
100	MHS DL Submit Permission Syntax
101	MHS O/R Address Syntax
102	MHS O/R Name Syntax
103	MHS Preferred Delivery Method Syntax
1000	Search Guide Syntax
1001	Postal Address Syntax
1002	Telex Number Syntax
1003	Teletex Terminal Identifier Syntax
1004	Fax Number Syntax
1005	Preferred Delivery Method Syntax
1006	Presentation Address Syntax
1007	Certificate Syntax
1008	Certificate Pair Syntax
1009	Certificate List Syntax
1010	Country Name Syntax
1011	Password Syntax
10000	Access Control List Syntax
10001	Shadowed By Syntax

Table G-2 Syntax

Digit	Explanation
0	No phonetic matching
1	Phonetic matching

Table G-3 Phonetic Flags

Digit	Explanation
0	Public
1	Standard
2	Sensitive

Table G-4 Access Classes

Digit	Explanation
0	Unlimited number of values
>0	Upper bound for number of values

Table G-5 Maximum Number of Values

Appendix H

XDS/XOM Header Files

This appendix lists the header files used for the DCE extensions to XDS/XOM: **<xdsgds.h>**, **<xdscds.h>**, **<xdsext.h>** and **<xomext.h>**. For information on the X/Open standard header files refer to:

- the **XDS** specification
- the **XOM** specification
- the **X.400** specification.

H.1 <xdsgds.h>

The **<xdsgds.h>** header declares the object identifiers of directory attribute types and directory object classes supported by the GDS Package (GDSP). It also defines OM classes used to represent the values of the attribute types.

All application programs that include this header must first include the **<xom.h>** header and the **<xds.h>** header.

```
#ifndef _XDSGDS_H
#define _XDSGDS_H

#ifndef XDSGDS_HEADER
#define XDSGDS_HEADER

/* GDS package object identifier */
/* iso(1) identified-organization(3) icd-ecma(0012)
   member-company(2) siemens-units(1107) sni(1) directory(3)
   xds-api(100)gdsp(0) */

#define OMP_O_DSX_GDS_PKG   \
"\x2B\x0C\x02\x88\x53\x01\x03\x64\x00"

/*Intermediate object identifier macros */

/* iso(1) identified-organization(3) icd-ecma(0012)
   member-company(2) siemens-units(1107) sni(1) directory(3)
   attribute-type(4) ...*/

#define dsP_GDSattributeType(X) \
("\x2B\x0C\x02\x88\x53\x01\x03\x04" #X)

/* iso(1) identified-organization(3) icd-ecma(0012)
   member-company(2) siemens-units(1107) sni(1) directory(3)
   object-class(6) ...*/

#define dsP_GDSobjectClass(X) \
("\x2b\x0c\x02\x88\x53\x01\x03\x06" #X)

#define dsP_gdsp_c(X)    OMP_O_DSX_GDS_PKG #X

/* OM class names (prefixed by DSX_C_)
   Directory attribute types (prefixed by DSX_A_)
   Directory object classes (prefixed by DSX_O_)
*/

/* Directory attribute types */

#define OMP_O_DSX_A_MASTER_KNOWLEDGE dsP_GDSattributeType(\x00)
#define OMP_O_DSX_A_ACL              dsP_GDSattributeType(\x01)
#define OMP_O_DSX_A_TIME_STAMP       dsP_GDSattributeType(\x02)
#define OMP_O_DSX_A_SHADOWED_BY      dsP_GDSattributeType(\x03)
#define OMP_O_DSX_A_SRT              dsP_GDSattributeType(\x04)
#define OMP_O_DSX_A_OCT              dsP_GDSattributeType(\x05)
```

```
#define OMP_O_DSX_A_AT                 dsP_GDSattributeType(\x06)
#define OMP_O_DSX_A_DEFAULT_DSA        dsP_GDSattributeType(\x08)
#define OMP_O_DSX_A_LOCAL_DSA          dsP_GDSattributeType(\x09)
#define OMP_O_DSX_A_CLIENT             dsP_GDSattributeType(\x0A)
#define OMP_O_DSX_A_DNLIST             dsP_GDSattributeType(\x0B)
#define OMP_O_DSX_A_SHADOWING_JOB      dsP_GDSattributeType(\x0C)
#define OMP_O_DSX_A_CDS_CELL           dsP_GDSattributeType(\x0D)
#define OMP_O_DSX_A_CDS_REPLICA        dsP_GDSattributeType(\x0E)

/* Directory object classes */

#define OMP_O_DSX_O_SCHEMA             dsP_GDSobjectClass(\x00)

/* OM class names */

#define OMP_O_DSX_C_GDS_SESSION           dsP_gdsp_c(\x00)
#define OMP_O_DSX_C_GDS_CONTEXT           dsP_gdsp_c(\x01)
#define OMP_O_DSX_C_GDS_ACL               dsP_gdsp_c(\x02)
#define OMP_O_DSX_C_GDS_ACL_ITEM          dsP_gdsp_c(\x03)

/* OM attribute names */

#define DSX_PASSWORD                      ((OM_type) 850)
#define DSX_DIR_ID                        ((OM_type) 851)
#define DSX_DUAFIRST                      ((OM_type) 852)
#define DSX_DONT_STORE                    ((OM_type) 853)
#define DSX_NORMAL_CLASS                  ((OM_type) 854)
#define DSX_PRIV_CLASS                    ((OM_type) 855)
#define DSX_RESIDENT_CLASS                ((OM_type) 856)
#define DSX_USEDSA                        ((OM_type) 857)
#define DSX_DUA_CACHE                     ((OM_type) 858)
#define DSX_MODIFY_PUBLIC                 ((OM_type) 859)
#define DSX_READ_STANDARD                 ((OM_type) 860)
#define DSX_MODIFY_STANDARD               ((OM_type) 861)
#define DSX_READ_SENSITIVE                ((OM_type) 862)
#define DSX_MODIFY_SENSITIVE              ((OM_type) 863)
#define DSX_INTERPRETATION                ((OM_type) 864)
#define DSX_USER                          ((OM_type) 865)
#define DSX_PREFER_ADM_FUNCS              ((OM_type) 866)
#define DSX_AUTH_MECHANISM                ((OM_type) 867)
#define DSX_AUTH_INFO                     ((OM_type) 868)
#define DSX_SIGN_MECHANISM                ((OM_type) 869)
#define DSX_PROT_REQUEST                  ((OM_type) 870)

/* DSX_Interpretation */

enum DSX_Interpretation {
        DSX_SINGLE_OBJECT    = 0,
        DSX_ROOT_OF_SUBTREE  = 1
};
```

```
/* DSX_Auth_Mechanism */

enum DSX_Auth_Mechanism {
        DSX_NONE_AT_ALL  = 0,
        DSX_DEFAULT      = 1,
        DSX_SIMPLE       = 2,
        DSX_SIMPLE_PROT1 = 3,
        DSX_SIMPLE_PROT2 = 4,
        DSX_DCE_AUTH     = 5,
        DSX_STRONG       = 6
};

/* DSX_Prot_Request */

enum DSX_Prot_Request {
        DSX_NONE   = 0,
        DSX_SIGNED = 1
};

/* upper bound on string lengths*/

#define DSX_VL_PASSWORD         ((OM_value_length) 16)

#endif  /* XDSGDS_HEADER */

#endif  /* _XDS_GDS_H */
```

H.2 <xdscds.h>

The **<xdscds.h>** header declares the object identifiers of directory attribute types supported by the Cell Directory Service (CDS).

All application programs that include this header must first include the **<xom.h>** header and the **<xds.h>** header.

```
#ifndef _XDSCDS_H
#define _XDSCDS_H

#ifndef XDSCDS_HEADER
#define XDSCDS_HEADER

/* {iso(1) identified-org(3) osf(22) dce(1) cds(3)
    = "\x2B\x16\x01\x03" */

/* Cell Directory Service attribute types */

#define OMP_O_DSX_A_CDS_Members          "\x2B\x16\x01\x03\x0A"
#define OMP_O_DSX_A_CDS_GroupRevoke      "\x2B\x16\x01\x03\x0B"
#define OMP_O_DSX_A_CDS_CTS              "\x2B\x16\x01\x03\x0C"
#define OMP_O_DSX_A_CDS_UTS              "\x2B\x16\x01\x03\x0D"
#define OMP_O_DSX_A_CDS_Class            "\x2B\x16\x01\x03\x0F"
#define OMP_O_DSX_A_CDS_ClassVersion     "\x2B\x16\x01\x03\x10"
#define OMP_O_DSX_A_CDS_ObjectUUID       "\x2B\x16\x01\x03\x11"
#define OMP_O_DSX_A_CDS_Address          "\x2B\x16\x01\x03\x12"
#define OMP_O_DSX_A_CDS_Replicas         "\x2B\x16\x01\x03\x13"
#define OMP_O_DSX_A_CDS_AllUpTo          "\x2B\x16\x01\x03\x14"
#define OMP_O_DSX_A_CDS_Convergence      "\x2B\x16\x01\x03\x15"
#define OMP_O_DSX_A_CDS_InCHName         "\x2B\x16\x01\x03\x16"
#define OMP_O_DSX_A_CDS_ParentPointer    "\x2B\x16\x01\x03\x17"
#define OMP_O_DSX_A_CDS_DirectoryVersion "\x2B\x16\x01\x03\x18"
#define OMP_O_DSX_A_CDS_UpgradeTo        "\x2B\x16\x01\x03\x19"
#define OMP_O_DSX_A_CDS_LinkTarget       "\x2B\x16\x01\x03\x1B"
#define OMP_O_DSX_A_CDS_LinkTimeout      "\x2B\x16\x01\x03\x1C"
#define OMP_O_DSX_A_CDS_Towers           "\x2B\x16\x01\x03\x1E"
#define OMP_O_DSX_A_CDS_CHName           "\x2B\x16\x01\x03\x20"
#define OMP_O_DSX_A_CDS_CHLastAddress    "\x2B\x16\x01\x03\x22"
#define OMP_O_DSX_A_CDS_CHUpPointers     "\x2B\x16\x01\x03\x23"
#define OMP_O_DSX_A_CDS_CHState          "\x2B\x16\x01\x03\x24"

/* {iso(1) identified-org(3) osf(22) dce(1) gds(2)
    = "\x2B\x16\x01\x02" */

#define OMP_O_DSX_UUID                  "\x2B\x16\x01\x02\x01"
#define OMP_O_DSX_TYPELESS_RDN          "\x2B\x16\x01\x02\x02"

#define OMP_O_DSX_NORMAL_SIMPLE_NAME    "\x2B\x16\x01\x03\x00"
#define OMP_O_DSX_BINARY_SIMPLE_NAME    "\x2B\x16\x01\x03\x02"

#endif  /* XDSCDS_HEADER */
#endif  /* _XDSCDS_H */
```

H.3 <xdsext.h>

The **<xdsext.h>** header, along with the **<xomext.h>** header, contain all the defines and function prototypes for the XDS / XOM convenience routines.

All application programs that include this header must first include the **<xom.h>**, **<xds.h>**, **<xdsgds.h>**, and **<xdscds.h> headers.**

```
#ifndef _XDSEXT_H
#define _XDSEXT_H

/*-- Function Prototypes ------------------------------------------*/

OM_return_code
dsX_extract_attr_values(
 OM_private_object      object,          /* IN-The source object           */
 OM_object_identifier   attribute_type,  /* IN-Attribute to be extracted   */
 OM_boolean             local_strings,   /* IN-Local strings required      */
 OM_public_object      *values,          /* OUT-Extracted Attribute Values */
 OM_value_position     *total_number);   /* OUT-Number of extracted values */

#endif  /* ifndef _XDSEXT_H */
```

H.4 <xomext.h>

The **<xomext.h>** header, along with the **<xdsext.h>** header, contain all the defines and function prototypes for the XDS / XOM convenience routines.

All application programs that include this header must first include the **<xom.h>**, **<xds.h>**, **<xdsgds.h>**, and **<xdscds.h> headers.**

```
#ifndef _XOMEXT_H
#define _XOMEXT_H

/*-- Defines for error returns related to XOI -----------------------  */

#define OMX_SUCCESS                 ((OM_integer)  0)
#define OMX_CANNOT_READ_SCHEMA      ((OM_integer) -1)
#define OMX_SCHEMA_NOT_READ         ((OM_integer) -2)
#define OMX_NO_START_OBJ_BLOCK      ((OM_integer) -3)
#define OMX_NO_END_OBJ_BLOCK        ((OM_integer) -4)
#define OMX_EMPTY_OBJ_BLOCK         ((OM_integer) -5)
#define OMX_OBJ_FORMAT_ERROR        ((OM_integer) -6)
#define OMX_DUPLICATE_OBJ_ABBRV     ((OM_integer) -7)
#define OMX_DUPLICATE_OBJ_OBJ_ID    ((OM_integer) -8)
#define OMX_NO_START_ATTR_BLOCK     ((OM_integer) -9)
#define OMX_NO_END_ATTR_BLOCK       ((OM_integer) -10)
#define OMX_EMPTY_ATTR_BLOCK        ((OM_integer) -11)
#define OMX_ATTR_FORMAT_ERROR       ((OM_integer) -12)
#define OMX_DUPLICATE_ATTR_ABBRV    ((OM_integer) -13)
#define OMX_DUPLICATE_ATTR_OBJ_ID   ((OM_integer) -14)
#define OMX_NO_START_CLASS_BLOCK    ((OM_integer) -15)
#define OMX_NO_END_CLASS_BLOCK      ((OM_integer) -16)
#define OMX_EMPTY_CLASS_BLOCK       ((OM_integer) -17)
#define OMX_CLASS_FORMAT_ERROR      ((OM_integer) -18)
#define OMX_NO_CLASS_NAME           ((OM_integer) -19)
#define OMX_DUPLICATE_CLASS_BLOCK   ((OM_integer) -20)
#define OMX_CLASS_BLOCK_UNDEFINED   ((OM_integer) -21)
#define OMX_INVALID_ABBRV           ((OM_integer) -22)
#define OMX_INVALID_OBJ_ID          ((OM_integer) -23)
#define OMX_INVALID_CLASS_NAME      ((OM_integer) -24)
#define OMX_INVALID_SYNTAX          ((OM_integer) -25)
#define OMX_MEMORY_INSUFFICIENT     ((OM_integer) -26)
#define OMX_INVALID_PARAMETER       ((OM_integer) -27)
#define OMX_UNKNOWN_ABBRV           ((OM_integer) -28)
#define OMX_UNKNOWN_OBJ_ID          ((OM_integer) -29)
#define OMX_UNKNOWN_OMTYPE          ((OM_integer) -30)

/*-- Defines for error returns related to convenience library ----------*/

#define OMX_MISSING_AVA             ((OM_integer) -101)
#define OMX_MISSING_ABBRV           ((OM_integer) -102)
#define OMX_FORMAT_ERROR            ((OM_integer) -103)
#define OMX_UNKNOWN_ERROR           ((OM_integer) -104)
#define OMX_MISSING_RDN_DELIMITER   ((OM_integer) -105)
#define OMX_MISMATCHED_QUOTES       ((OM_integer) -106)
#define OMX_MISSING_EQUAL_OPERATOR  ((OM_integer) -107)
#define OMX_MISSING_ATTR_VALUE      ((OM_integer) -108)
#define OMX_MISSING_ATTR_INFO       ((OM_integer) -109)
#define OMX_MISSING_CLASS_START_OP  ((OM_integer) -110)
#define OMX_MISSING_CLASS_END_OP    ((OM_integer) -111)
#define OMX_MISSING_CLASS_VALUE     ((OM_integer) -112)
```

```
#define OMX_MISSING_COMP_VALUE      ((OM_integer) -113)
#define OMX_MISMATCHED_BRACKETS     ((OM_integer) -114)
#define OMX_UNEXPECTED_OPERATOR     ((OM_integer) -115)
#define OMX_WRONG_VALUE             ((OM_integer) -116)
#define OMX_UNKNOWN_KEYWORD         ((OM_integer) -117)
#define OMX_MISSING_OPERATOR        ((OM_integer) -118)
#define OMX_MISSING_COMPOUND_OP     ((OM_integer) -119)

/*- Additional Errors returned by the omX_object_to_string function -*/

#define OMX_CLASS_NOT_FOUND_IN_SCHEMA_FILE  ((OM_return_code) 31)

/*-- Function Prototypes ------------------------------------------*/

OM_return_code omX_fill(
 OM_type           type,         /* IN  - Type of Object           */
 OM_syntax         syntax,       /* IN  - Syntax of the object     */
 OM_uint32         length,       /* IN  - Data length              */
 void             *elements,     /* IN  - Data Value               */
 OM_descriptor    *destination); /* OUT - The filled up descriptor */

OM_return_code omX_fill_oid(
 OM_type               type,         /* IN  - Type of Object           */
 OM_object_identifier  object_id,    /* IN  - Value of the object      */
 OM_descriptor        *destination); /* OUT - The filled up descriptor */

OM_return_code omX_extract(
 OM_private_object  object,           /* IN  - Extract from this object    */
 OM_type_list       navigation_path,  /* IN  - Leads to the target object  */
 OM_exclusions      exclusions,       /* IN  - Scope of extraction         */
 OM_type_list       included_types,   /* IN  - Objects to be extracted     */
 OM_boolean         local_strings,    /* IN  - Local strings required      */
 OM_value_position  initial_value,    /* IN  - First value to be extracted */
 OM_value_position  limiting_value,   /* IN  - Last value to be extracted  */
 OM_public_object  *values,           /* OUT - Array of extracted objects  */
 OM_value_position *total_number);    /* OUT - Count of extracted objects  */

OM_return_code omX_string_to_object(
 OM_workspace          workspace,     /* IN  - The workspace               */
 OM_string            *string,        /* IN  - The string to be converted  */
 OM_object_identifier  class,         /* IN  - The OM Class to be created  */
 OM_boolean            local_strings, /* IN  - Local strings specified     */
 OM_private_object    *object,        /* OUT - The converted Object        */
 OM_integer           *error_position,/* OUT - Error Position in I/P string*/
 OM_integer           *error_type);   /* OUT - Type of error               */

OM_return_code omX_object_to_string(
    OM_object    object,        /* The Object to be converted          */
    OM_boolean   local_strings, /* To indicate local string conversion */
    OM_string   *string);       /* The converted DN string             */

#endif  /* ifndef _XOMEXT_H */
```

Index