X/Open CAE Specification

API to Directory Services (XDS)

Issue 3

X/Open Company Ltd.

X/Open CAE Specification

API to Directory Services (XDS) Issue 3

ISBN: 1-85912-180-2

X/Open Document Number: C608

Published by X/Open Company Ltd., U.K.

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

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.org

Contents

List of Figures

List of Tables

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done in any one of the following ways:

- anonymous ftp to ftp.xopen.org
- ftpmail (see below)
- reference to the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information using ftpmail, send a message to ftpmail@xopen.org with the following four lines in the body of the message:

```
open
cd pub/Corrigenda
get index
quit
```

This will return the index of publications for which Corrigenda exist. Use the same email address to request a copy of the full corrigendum information following the email instructions.

This Document

This document is a CAE Specification (see above). It defines the application programming interface (API) to X/Open Directory Services. This interface is designed to offer services that are consistent with, but not limited to, the 1988 CCITT X.500-Series of Recommendations and the ISO 9594 Standard.

This *Issue 3* of the XDS CAE Specification includes revisions to align it with the ISO DIS 14392/3/4/5 Directory Services group of Standards, which themselves are based on the corresponding IEEE DS Standards, which in turn were developed from the original CAE version (C190) of the X/Open XDS specification.

All new implementation work by API providers should be based on XDS Issue 3.

X/Open did publish an IEEE-aligned XDS Issue 2 (C317) specification. However, XDS Issue 2 is now deprecated in favour of the ISO-aligned XDS Issue 3 specification.

This is one of several specifications that X/Open originally developed in collaboration with the XAPI Association. The other documents are OSI Abstract Data Manipulation (XOM), Electronic Mail (X.400), Message Store (XMS) and EDI, all API specifications. In addition, X/Open has published a Guide to Selected X.400 and Directory Services APIs.

The XOM and XMHS API specifications have similarly served as bases for corresponding IEEE and ISO Standards. X/Open has also revised its XOM and XMHS specifications into *Issue 3* publications, to align them with the corresponding ISO Standards.

Structure

This document is organised as follows:

- Chapter 1 identifies the scope and purpose of this API, gives a brief introduction to Directory concepts, and introduces Object Management, which is a data-handling API used as part of this interface.
- Chapter 2 describes the C language binding provided in this document.
- Chapter 3 provides a general description of the interface.
- Chapter 4 specifies the interface functions.
- Chapter 5 defines the OM classes that constitute the Directory Service (DS) package.
- Chapter 6 defines the Errors that can arise in the use of the interface, and the method used to report them, for the DS package.
- Chapter 7 defines directory class definitions for three further packages:
  - — the Basic Directory Contents Package (BDCP)
  - — the Strong Authentication Package (SAP)
  - — the MHS (Message Handling System) Directory User Package (MDUP).
- Chapter 8 sets out the symbols used in the header files for the DS, BDCP, SAP and MDUP packages.
- Appendix A provides some programming examples.
- Appendix B identifies the few divergences that exist between the IEEE Directory Services Standard and the X/Open XDS Issue 2 CAE Specification.
- Appendix C identifies the known substantive differences between the X/Open ISO-Standard-aligned XDS CAE API Specification C608 and the corresponding ISO XDS Standard.
- Appendix D identifies the known substantive differences between the X/Open ISO-Standard-aligned XDS CAE API Specification C608 and the original XDS CAE API Specification C190 (referred to in existing XMHS brands).

A glossary and index are provided.

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, and language-independent names.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — command operands, command option-arguments or 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*( ). Names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.

- Roman font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
- The notation [EABCD] is used to identify a return value ABCD, including if this is an an error value.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items.
- For a more detailed description of the C language binding font usage, see Chapter 2.

Trade Marks

X/Open® is a registered trade mark, and the ‘‘X’’ device is a trade mark, of X/Open Company Limited.

Referenced Documents

The following documents are referenced in this specification:

ANSI-C
: Information Processing: Programming Language C, ISO Draft International Standard DIS9899 (also known as *ANSI C*, American National Standard X3.159-1989)

ASN.1
: Specification of Abstract Syntax Notation One (ASN.1), ISO 8824 (including Addendum 1), CCITT[1] X.208

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).

IEEE 1224.2-1993
: IEEE 1224.2-1993: IEEE Standard for Information Technology - Open Systems Interconnection (OSI) Directory Services - Application Programming Interface (API) [Language Independent], ISBN 1-55937- - .

IEEE 1327.2-1993
: IEEE 1327.2-1003: IEEE Standard for Information Technology - Open Systems Interconnection (OSI) Directory Services C Language Interfaces - Binding for Application Programming Interface (API), ISBN 1-55937- - .

Fascicle VII.3
: Fascicle VII.3, Terminal Equipment and Protocols for Telematic Services, October 1984

ISO 9594-1
: Information Technology - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Services, ISO 9594-1, CCITT X.500

ISO 9594-2
: Information Technology - Open Systems Interconnection - The Directory: Models, ISO 9594-2, CCITT X.501

ISO 9594-3
: Information Technology - Open Systems Interconnection - The Directory: Abstract Service Definition, ISO 9594-3, CCITT X.511

ISO 9594-6
: Information Technology - Open Systems Interconnection - The Directory: Selected Attribute Types, ISO 9594-6, CCITT X.520

---

1. The CCITT organization has now been re-named as the Telecommunications Standardization Section of the International Telecommunications Union (ITU-T). However, to retain the historical perspective of those references which were developed when it was known as CCITT, this name has not been replaced here.

ISO 9594-7
Information Technology - Open Systems Interconnection - The Directory: Selected Object Classes, ISO 9594-7, CCITT X.521

ISO 9594-8
Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, ISO 9594-8, CCITT X.509

ISO 8824
ISO 8824: 1990 (CCITT X.208: 1988), Information Technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1),

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

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

XPG4
X/Open Systems and Branded Products: XPG4, October 1994 (ISBN: 1-872630-52-9, X924).

The following are the IEEE **Directory Services** Standards:

IEEE 1224.2-1993:
Directory Services, Language-Independent Standard.

IEEE 1327.2-1993:
Directory Services, C Binding.

The following are the ISO **Directory Services** Standards:

ISO DIS 14392:
Directory Services, Language-Independent Specification

ISO DIS 14394:
Directory Services, C Language Binding.

Chapter 1

Introduction

Note: Many technical terms, such as *object* and *attribute*, are used by both object management and the directory. The meanings ascribed to these terms are often similar but different. (See Section 1.4) and the Glossary provided at the end of this document.

1.1 Overview

The X/Open Directory Services Application Program Interface (abbreviated XDS) defines an Application Program Interface (API) to directory services in the X/Open Common Applications Environment (CAE) defined in the **X/Open Portability Guide** (see **Referenced Documents**). It is referred to as *the interface* throughout this specification.

This interface is designed to offer services that are consistent with, but not limited to, the 1988[1] CCITT X.500-Series of Recommendations and the ISO 9594 Standard. The CCITT Recommendations and the ISO Standard were developed in close collaboration and are technically aligned. They are referred to as **the standards** throughout this specification. Access to other directory services through the API is not prohibited, but has not been explicitly considered.

The interface is designed for operational interactions with a directory, rather than for management interactions such as knowledge management or schema management. Also, the security features of the standards are not generally visible in the interface, in order to permit flexibility in security policies.

The directory is a distributed collection of information, which programs can access through the interface in order to make queries or updates. A brief introduction to directory concepts is given in Section 1.3. Section 1.4 gives an overview of object management, which is a data-handling API used as a part of the interface. Section 1.5 on page 10 describes the optional features of this specification, and finally Section 1.6 gives a list of abbreviations used. In all cases, the reader should refer to the standards or to the referenced **XOM** Specification (see **Referenced Documents**) for further details.

1. It also takes account of some changes made in the 1992 version of these standards.

1.2 Format of the Specification

This specification describes a programming language-independent interface to the directory together with a specific C language binding of that interface. Several typographic conventions are used to identify particular items. The general conventions are described in the Preface, while the C language binding conventions are described in Chapter 2.

1.3 The Directory

The interface is based on a model of the directory that is described in the standards, and it provides facilities that closely follow those described there.

The directory is a collection of open systems that cooperate to hold information about *objects* in the real world. Directory *users*, including people and programs, can read or modify this information. The information is typically used to facilitate communication between objects such as people, application programs, terminals and so on.

The directory holds information in the *Directory Information Base* (DIB), with an *entry* for each object. The entries are made up of *attributes*, which each have a *type* and one or more *values*, describing things such as names, telephone numbers and addresses as required.

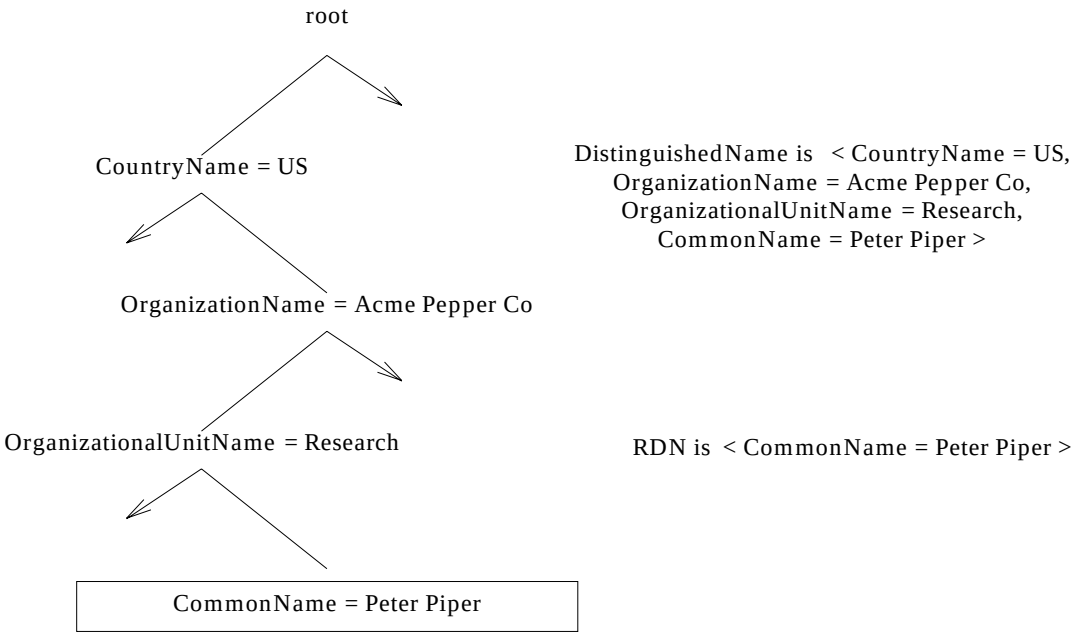

Figure 1-1 Example Directory Distinguished Name

The DIB is structured into a tree, called the *Directory Information Tree* (DIT), with the entries at the nodes of the tree. The position of each entry in the tree is fixed by its *distinguished name*, which is formed by appending its own *relative distinguished name* (RDN) to the distinguished name of the immediately superior entry in the tree. The root of the tree has an empty name. The RDN of an entry is some selection of its attribute values, chosen to identify it uniquely. Clearly, the RDN of each entry must be different from all other immediate subordinates of its immediate superior, so that its name is unambiguous. Names are discussed in Section 3.5.3 on page 21.

Further structure is provided in the directory by constraining the attributes of entries at each particular part of the tree. This is done by means of *Object Classes*, which contain lists of mandatory and optional attributes, and by structure rules, which set down the permissible object class of entries at every place in the tree. Each entry has an attribute of type **Object Class**, whose values are the object classes to which it belongs.

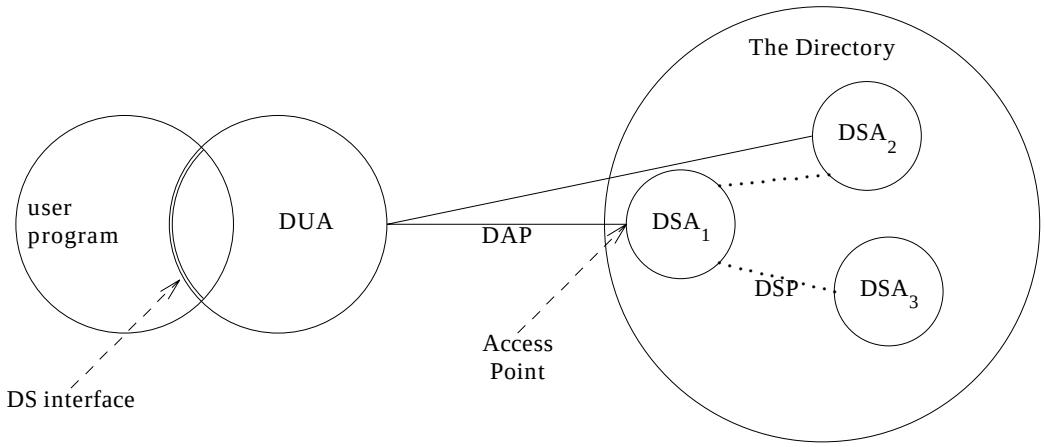

Figure 1-2 Overview of Directory Components

The directory is defined in terms of *Directory System Agents* (DSAs) and *Directory User Agents* (DUAs). Loosely, the DSAs correspond to individual computers each holding a part of the DIB containing the entries for a group of objects, while the DUAs represent the user in interactions with the DSAs. The standards define the service used by DUAs to query and change the directory contents; it is called the Directory Service or the Abstract Service. This defines the operations such as **Read** and **Modify-Entry**. A *Directory Access Protocol* (DAP) is also defined, which a DUA can use to obtain this service from any DSA.

The DS interface defined in this specification is the interface between a DUA and an application program (which is the *user*), by which the application program can access the Directory Service.

Where a DSA cannot provide service to a DUA because the required information is held elsewhere, the DSA may also interact with other DSAs in order to provide the service. The standards set out how this can be done using a *Directory System Protocol* (DSP). The DSA may instead choose just to inform the DUA, or the calling DSA, where the required information can be found. This is called a *referral* and may occur because of the user's preferences or the DSA's circumstances.

1.4 Object Management

The interface makes use of facilities provided by the OSI Object Management API (OM API). These facilities are fully described in the referenced **XOM** Specification, and are introduced briefly below.

Throughout this document, care is taken to distinguish between OM classes and directory classes, and between OM attributes and directory attributes. In both cases, the former is a construct of the object management interface, while the latter is a construct of the directory service to which the interface provides access. The unqualified term *attribute* denotes the directory construct, while the phrase *OM attribute* denotes the object management construct. The phrase *Object Class* denotes the directory construct, while the phrase *OM class* denotes the object management construct.

The subsections below introduce the various concepts that are used in object management, starting with the smallest.

1.4.1 Syntax

A *syntax* is the basis for the classification and representation of values in object management. Examples of syntaxes are Boolean, Integer, String(Octet) and Object.

Syntaxes are defined in the Object Management Specification, and nowhere else, and are themselves represented by integers.

1.4.2 Value

A *value* is a single datum, or piece of information. Each value belongs to exactly one syntax by which its representation is defined. A value may be as simple as a Boolean value (for example, **true**), or as complicated as an entire OM object (for example, a Message).

1.4.3 OM Attribute

An *OM attribute type* is an arbitrary category into which a specification places some values.

OM attribute types are represented by integers, which are assigned in individual service specifications, and which are only meaningful within a particular package (see Section 1.4.6 on page 7.

An *OM attribute* is an OM attribute type, together with an ordered sequence of one or more values. OM attributes can occur only as parts of an OM object and the OM attribute type, and values are constrained by the OM class specification of that OM object (see Section 1.4.5 on page 6.

The OM attribute type can be thought of as the name of the OM attribute.

There is no general representation for an OM attribute, but a descriptor represents an OM attribute type together with a single syntax and value (see Section 1.4.9 on page 8).

1.4.4 OM Object

An *OM object* is a collection of OM attributes, the values of which can be accessed by means of functions. The particular OM attribute types that may occur in an OM object are determined by the OM class of the OM object (see Section 1.4.5), as are the constraints on those OM attributes. The OM class of an OM object is determined when the OM object is created, and cannot be changed.

OM objects are represented in the interface by a handle, or opaque pointer. The internal representation of an OM object is not specified, though there is a defined data structure, called a *descriptor list*, which can also be used directly in a program (see Section 1.4.9 on page 8).

1.4.5 OM Class

An *OM class* is a category of OM object, set out in a specification. It determines the OM attributes that may be present in the OM object, and details the constraints on those OM attributes.

Each OM object belongs directly to exactly one OM class, and is called an *instance* of that OM class.

The OM classes of OM objects form a tree; each OM class has exactly one immediate *superclass* (except for the OM class *Object*, which is the root of the tree), and each OM class may have an arbitrary number of *subclasses*. The tree structure is also known as the *OM class hierarchy*. The importance of the OM class hierarchy stems from the inheritance property discussed below.

Each OM class of OM object has a fixed list of OM attribute types, and every OM object that is an instance of the OM class has only these OM attributes (some OM attributes may not be present in particular instances, as permitted by the constraints in the OM class specification). The list of OM attribute types that may appear in instances of an OM class has two parts. Each OM class *inherits* all the OM attribute types that are permitted in its immediate superclass as legal OM attribute types. There is also a list of additional OM attribute types that are permitted in the OM class. Any subclasses of this OM class will inherit all of these OM attribute types, from both lists.

Because of inheritance, an OM object is also said to be an instance of all its superclasses. It is required that the OM class constraints of each superclass are met, considering just those OM attribute types that are permitted in the superclass.

The OM class hierarchy and the list of OM attribute types for each OM class are determined solely by the interface specification and cannot be changed by a program.

The OM class specification may impose arbitrary constraints on the OM attributes. The most common of these are tabulated in the OM class specification and are marked with a * below. Frequently encountered cases include constraints as follows:

- to restrict the syntaxes permitted for values of an OM attribute (often to a single syntax) *
- to restrict the particular values to a subset of those permitted by the syntax
- to require one or more values of the OM attribute (a *mandatory* OM attribute) *
- to require either zero or more values of the OM attribute (an *optional* OM attribute) *
- to permit multiple values, perhaps up to some limit known as the *value number constraint* *
- to restrict the length of strings, up to a limit known as the *value length constraint* *.

  Note: The constraint is expressed in terms of bits, octets or characters according to the kind of string. However, the lengths of strings are stated everywhere else in terms

of the number of *elements*, which are either bits or octets. The number of elements in a string with multibyte characters (for example, T.61 Teletext) may thus exceed the value length constraint. (In C, an array with more bytes will be needed to store it.)

The constraints may affect multiple OM attributes at once, for example a rule that only one of several OM attributes may be present in any OM object.

Every OM object includes the OM class to which it belongs as the single value of the mandatory OM attribute type **Class**, which cannot be modified. The value of this OM attribute is an OSI Object Identifier, which is assigned to the OM class by the specification.

An *abstract class* is an OM class of which instances are forbidden. It may be defined as a superclass in order to share OM attributes between OM classes, or simply to ensure that the OM class hierarchy is convenient for the interface definition.

1.4.6 Package

A *Package* is a set of OM classes that are grouped together by the specification, because they are functionally related.

A package is identified by an OSI Object-Identifier, which is assigned to the package by the specification. Thus the identity of each package is completely unique.

1.4.7 Package Closure

An OM class may be defined to have an OM attribute whose value is an OM object of an OM class defined in some other package. This is done to share definitions and to avoid duplication. For example, the Directory Contents package defined in Chapter 7 defines an OM class called **Teletex-Terminal-Identifier**. This OM class has an OM attribute whose value is an OM object of OM class **Teletex-NBPs**, that is defined in the Message Transfer package in the referenced **X.400** Specification. An OM class may also be a subclass of an OM class in another package. These relationships between packages lead to the concept of a Package-Closure.

A *Package-Closure* is the set of classes which need to be supported in order to be able to create all possible instances of all classes defined in the package. (A formal definition is given in the referenced **XOM** Specification.)

1.4.8 Workspace

Details of the representation of OM objects, and of the implementation of the functions that are used to manipulate them, are not specified because they are not the concern of the application programmer. However, the programmer sometimes needs to be aware of which implementation is being used for a particular OM object.

The case of the OM class **Teletex-NBPs** was mentioned above. This OM class is used in both the Message Transfer Service and in the Directory Service. If an application uses both services, and the two services use different internal representations of OM objects (perhaps because they are supplied by different vendors), then it is necessary for the application to specify which implementation should create a **Teletex-NBPs** OM object. This is done by means of a workspace.

A *workspace* is one or more Package-Closures, together with an implementation of the object management functions that supports all the OM classes of OM objects in the Package-Closures.

The notion of a workspace also includes the storage used to represent OM objects and management of that storage. The interested reader can refer to Chapter 5 of the referenced **XOM** Specification for more details of how workspaces are implemented.

The application must obtain a workspace that supports an OM class before it is able to create any OM objects of that OM class. The workspaces are returned by functions in the appropriate service. For example, *DS-Initialize*( ) returns a workspace that supports the Directory Service package, whilst another function in another OSI service returns a workspace that supports another package.

Some implementations may support additional packages in a workspace. For example, vendors may provide OM classes for directory attribute types in the Basic Directory Contents and the Strong Authentication Packages (*DS-Version*( )) may be used to negotiate these packages into a workspace obtained from *DS-Initialize*( ). Another important case is where two or more services are supported by the same implementation. In this case, the workspaces returned by *DS-Initialize*( ) and *MT-Open*( ) (see the referenced **X.400** Specification) are likely to have the same implementation. The application need take no account of this, but may experience improved performance.

1.4.9 Descriptor

A *descriptor* is a defined data structure that is used to represent an OM attribute type and a single value. The structure has three components: a type, a syntax and a value.

A *descriptor list* is an ordered sequence of descriptors that is used to represent several OM attribute types and values.

Where the list contains several descriptors with the same OM attribute type (representing a multi-valued OM attribute), the order of the values in the OM attribute is the same as the order in the list. Such descriptors will always be adjacent.

Where the list contains a descriptor representing the OM class, this must occur before any others.

A *public object* is a descriptor list that contains all the OM attribute values of an OM object, including the OM class. Public objects are used to simplify application programs by enabling the use of static data structures instead of a sequence of OM function calls, as can be seen in Appendix A.

A *private object* is an OM object created in a workspace using the object management functions or the functions in an OSI service. The term is simply used for contrast with a public object.

1.4.10 Use of Objects

OM objects are used to represent the data collections used in the interface, such as *directory entry* attributes and directory operation results. Refer to

- **Interface Class Definitions** - see Chapter 5
- **Errors** - see Chapter 6
- **Directory Class Definitions** - see Chapter 7

for the definition of these OM classes.

An important feature of the interface is that an instance of a subclass can be used wherever a particular OM class is needed. This means both that the application can supply a subclass and that the service can return a subclass. For example, the application can supply names in any format that is defined as a subclass of the abstract class *Name*, and the Directory Service returns all errors in subclasses of the abstract class *Error*.

Because the service may return a subclass of the specified OM class, applications should always use the *OM-Instance*( ) function when checking the OM class of an OM object, rather than testing the value of the **Class** OM attribute.

When the application supplies a subclass of the specified OM class as an argument, the service will either recognise them as vendor extensions or will ignore all OM attribute types that are not permitted in the specified OM class.

The application can generally supply either a public object or a private object as an argument of the interface functions. There are exceptions such as the **Context** and **Session** arguments, which must be private objects in the interests of efficiency. The interface will always return private objects. The application can convert these into public objects by a call to *OM-Get*( ) if required.

Note: Note that public objects returned by *OM-Get*( ) are read-only and must not be modified in any way.

1.5 Mandatory and Optional Features

The interface defines an Application Program Interface (API) that application programs can use to access the functionality of the underlying directory service. The interface does *not* define or imply any profile of that service.

Note that nothing in this specification requires that the implementation of the interface, or the directory itself, actually make use of DAP, DSP or other parts of the model, as long as it provides the defined service. Also, the *scope* of the directory to which an application has access is not determined. It is entirely a local matter whether objects in other DSAs are accessible.

All the interface functions are mandatory, there are no optional ones.

Some OM attributes are optional; these are marked *(Optional Functionality)* in the OM class definitions. They are:

- **Asynchronous** in a **Context** object (and consequent availability of asynchronous operations)
- **File-Descriptor** in a **Session** object.

Some items of behaviour of the interface are implementation-defined. These are:

- the maximum number of outstanding asynchronous operations
- whether an asynchronous function call returns before the operation is submitted to the directory
- the text and language of error messages
- the provision, or not, of automatic connection management
- whether *Abandon*( ) issues an **Abandon** operation or not
- the OM classes permitted as values of the **Name** argument to interface functions.

The default values of several OM attributes in **Context** and **Session** OM objects are locally administered.

This specification defines four packages; one of which is mandatory and three are optional. Use of the optional packages is negotiated using the *Version*( ) function.

- The Directory Service package defined in Chapter 5, which also includes the errors defined in Chapter 6, is mandatory.
- The Basic Directory Contents package defined in Chapter 7 is optional.
- The Strong Authentication package, also defined in Chapter 7, is optional.
- The MHS Directory User package, also defined in Chapter 7, is optional.

This specification does not mandate that any OM classes are encodable using *OM-Encode*( ) and *OM-Decode*( ).

1.6 Terminology

The terms *implementation-defined, may, should, undefined, unspecified* and *will* are used in this document with the meanings ascribed to them in the Glossary providfed at the end of this document.

1.6.1 Abbreviations

API	Application Program Interface
ASN.1	Abstract Syntax Notation One
AVA	Attribute Value Assertion
BER	Basic Encoding Rules
CA	Certification Authority
CCITT	International Telegraph and Telephone Consultative Committee
DAP	Directory Access Protocol
DIB	Directory Information Base
DIT	Directory Information Tree
DMD	Directory Management Domain
DN	Distinguished Name
DS	Directory Service
DSA	Directory System Agent
DSP	Directory System Protocol
DUA	Directory User Agent
IA5	International Alphabet No. 5
ID	Identifier
ISDN	Integrated Services Digital Network
ISO	International Organisation for Standardisation
NBP	Non-Basic Parameter
MS	Message Store
OM	Object Management
OSI	Open Systems Interconnection
RDN	Relative Distinguished Name
ROSE	Remote Operations Service Element
XDS	X/Open Directory Services API.
XOM	X/Open OSI-Abstract-Data Manipulation API

1.7 Future Directions

This Section notes a number of areas where there are likely to be future developments in the interface.

- Future versions of this interface will provide access to the functionality specified in future versions of the standards.
- It is likely that additional representations of names will be adopted as such representations become widely accepted. In particular, it is likely that:
  - — a string representation of names will be included
  - — a technique for supplying names relative to a naming context will be added (*abbreviated names*).
- Further standardisation of the OM representation of directory attributes is likely. This will include the definition of new packages for additional attribute types and object classes, and may also include a means to dynamically extend the set of recognised attribute types.
- The definition of portable provision of security features will proceed. This may affect future versions of this interface, or may be provided by a separate interface.
- Future versions of the interface are expected to be compatible with this version.

Chapter 2

C Language Binding

2.1 Introduction

This Chapter sets out certain characteristics of the C language binding to the interface. The binding specifies C identifiers for all the elements of the interface, so that application programs written in C can access the directory. These elements include function names, typedef names and constants. All the C identifiers are mechanically derived from the language-independent names as explained below. There is a list of the identifiers in Chapter 8. For ease of use, some of these identifiers are defined in the specification alongside the language-independent name.

A *Function*( ) is indicated as shown.

A CONSTANT is in Roman font.

The names of [ERRORS] and other return codes are surrounded by square brackets.

The definitions of the C identifiers appear in five headers:

- **<xom.h>** contains definitions for the associated OM interface
- **<xds.h>** contains definitions for the Directory Service
- **<xdsbdcp.h>** contains definitions for the Basic Directory Contents Package attributes
- **<xdssap.h>** contains definitions for the Strong Authentication Package attributes
- **<xdsmdup.h>** contains definitions for the MHS Directory User Package attributes.

These header files are described in Chapter 8.

2.2 C Naming Conventions

The interface uses part of the C public namespace for its facilities. All identifiers start with the letters *ds*, *DS* or *OMP*, and more detail of the conventions used is given in the following table. Note that the interface reserves *all* identifiers starting with the letters *dsP* for Private (that is, internal) use by implementations of the interface. It also reserves *all* identifiers starting with the letters *dsX* or *DSX* for vendor-specific extensions of the interface. Application programmers should not use any identifier starting with these letters.

The Object Management API uses similar, though not identical, naming conventions, which are described in the referenced **XOM** Specification. All its identifiers are prefixed by the letters *OM* or *om*.

Item	**Prefix**
reserved for implementors	`dsP`
reserved for interface extensions	`dsX`
reserved for interface extensions	`DSX`
reserved for implementors	`OMP`
<xds.h>	
functions	`ds_`
error *problem* values	`DS_E_`
OM class names	`DS_C_`
OM value length limits	`DS_VL_`
OM value number limits	`DS_VN_`
other constants	`DS_`
<xdsbdcp.h> <xdssap.h> and <xdsmdup.h>	
Attribute Type	`DS_A_`
Object Class	`DS_O_`

Table 2-1 C Naming Conventions

A complete list of all identifiers used (except those beginning `dsP, dsX, DSX or OMP`) is given in Chapter 8. No implementation of the interface will use any other public identifiers. A *public identifier* is any name except those reserved in Section 4.1.2.1 of the ISO C Standard (see the referenced **ANSI C** document), and the *public namespace* is the set of all possible public identifiers.

The C identifiers are derived from the language-independent names used throughout this specification by a purely mechanical process which depends on the kind of name.

- Function names are made entirely lower-case and prefixed by *ds_*. Thus **Receive-result** becomes *ds_receive_result*( ).
- C function parameters are derived from the argument and result names by making them entirely lower-case. In addition the names of results have *_return* added as a suffix. Thus the argument *Name* becomes *name*, whilst the result **Operation-Status** becomes *operation_status_return*.
- OM class names are made entirely upper-case and prefixed by DS_C_. Thus **Read-Result** becomes DS_C_READ_RESULT .
- Enumeration tags are derived from the name of the corresponding OM syntax by prefixing with DS_ . The case of letters is left unchanged. Thus **Enum(Limit-Problem)** becomes DS_Limit_Problem .

- Enumeration constants, as well as the names of OM attributes and all other constants except errors, are made entirely upper-case and prefixed by DS_ . Thus **O-Residential-Person** becomes DS_O_RESIDENTIAL_PERSON .
- Errors are treated as a special case. Constants that are the possible values of the OM attribute *Problem* of a subclass of the OM class *Error* are made entirely upper-case and prefixed by DS_E_ .
  Thus **alias-dereferencing-problem** becomes DS_E_ALIAS_DEREFERENCING_PROBLEM.
- The constants in the **Value Length** and **Value Number** columns of the OM class definition tables are also assigned identifiers. (They have no names in the language-independent specification.) Where the upper limit in one of these columns is not '1' (one), it is given a name consisting of the OM attribute name, prefixed by DS_VL_ for value length, or DS_VN_ for value numbers.
- The sequence of octets for each object identifier is also assigned an identifier for internal use by certain OM macros. These identifiers are all upper case and are prefixed by OMP_O_. See the referenced **XOM** Specification for further details on the use of object identifiers.

Note that hyphens are translated everywhere to underscores.

2.3 Use and Implementation of Interfaces

Each of the following statements applies unless explicitly stated otherwise in the detailed descriptions that follow.

If an argument to a function has an invalid value (such as a value outside the domain of the function, a pointer outside the address space of the program, or a null pointer), the behaviour is undefined.

Any function declared in a header may be implemented as a macro defined in the header, so a library function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is not then followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a library function even if it is also defined as a macro. The use of **#undef** to remove any macro definition will also ensure that an actual function is referred to. Any invocation of a library function that is implemented as a macro will expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression anywhere a function with a compatible return type could be called.

2.4 Function Return Values

The return value of a C function is always bound to the **Status** result of the language-independent description. Functions return a value of type DS_status, which is an error indication. If, and only if, the function succeeds, its value will be **success**, expressed in C by the constant DS_SUCCESS. If a function returns a status other than this, then it has not updated the return parameters. The value of the status in this case is an error as defined in Chapter 6.

Since C does not provide multiple return values, functions must return all other results by writing into storage passed by the application program. Any argument that is a pointer to such storage has a name ending with *_return*. For example, the C parameter declaration *Uint * completion_flag_return* indicates that the function will return an unsigned integer **Completion-Flag** as a result, so the actual argument to the function must be the address of a suitable variable. This notation allows the reader to distinguish between an input parameter that happens to be a pointer, and an output parameter where the '*' is used to simulate the semantics of passing by reference.

Chapter 3

Description

3.1 Introduction

The interface comprises a number of functions, together with many 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 closely on the *Abstract Service* that is specified in the standards (see reference **ISO 9594-3**).

The interface models 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* within the directory, which eventually returns a *status* and any *result* of the operation.

All interactions between the user and the directory 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. The context includes a number of parameters that are common to many functions and that seldom change from operation to operation.

Each of the components of this model are described below, along with other features of the interface such as asynchronous function calls and security.

3.2 Services

As mentioned above, the standards define Abstract Services that requestors use to interact with the directory. Each of these Abstract Services maps to a single function call with the same name (with two minor exceptions noted below), and the detailed specifications are given in Chapter 4. The services are:

DirectoryBind (maps to *Bind*( ))
DirectoryUnbind (maps to *Unbind*( ))
Read
Compare
Abandon
List
Search
AddEntry
RemoveEntry
ModifyEntry
ModifyRDN

There is a function called *Receive-result*( ) which has no counterpart in the abstract service. It is used with asynchronous operations, and is explained in Section 3.7 on page 23. There are also additional functions called *Initialize*( ), *Shutdown*( ) and *Version*( ).

The interface functions are summarised in the table below. Those which can execute asynchronously are indicated by an *a* in the table. All other functions always execute synchronously.

	Name	Description
	Abandon	abandon the result of a pending, asynchronous operation.
a	**Add-Entry**	add a leaf entry to the directory information tree.
	Bind	open a session with a directory user agent.
a	**Compare**	compare a purported attribute value with the attribute value stored in the directory for a particular entry.
	Initialize	initialise the interface and allocate a workspace.
a	**List**	enumerate the immediate subordinates of a particular directory entry.
a	**Modify-Entry**	perform an atomic modification of a directory entry.
a	**Modify-RDN**	change the Relative Distinguished Name of a leaf entry.
a	**Read**	query information on a directory entry by name.
	Receive-Result	retrieve the result of an asynchronously executed function.
a	**Remove-Entry**	remove a leaf entry from the directory information tree.
a	**Search**	find entries of interest in a portion of the directory information tree.
	Shutdown	discard a workspace.
	Unbind	unbind from a directory session.
	Version	negotiate features of the interface and service.

Table 3-1 Interface Functions

3.2.1 Negotiation Sequence

The interface has an initialisation and shutdown sequence that permits the negotiation of optional features. This involves the functions *Initialize*( ), *Version*( ) and *Shutdown*( ).

Every application program must first call *Initialize*( ), which returns a workspace. This workspace supports the standard Directory Service package (see Chapter 5, and no other packages or extensions that affect the behaviour of applications, making use only of standard features. In particular, this means that any attribute values returned by the service will be represented as an object of OM class *Encoding*, as discussed in Section 3.5.1 on page 20.

The workspace can be extended to support the optional packages Basic Directory Contents, Strong Authentication, and or MHS Directory User, (see Chapter 7, or any vendor extensions. Vendor extensions may include additional packages, and may also include additional or modified functionality. All such packages or other extensions are identified by means of OSI Object Identifiers, and these Object Identifiers are supplied to the *Version*( ) function to incorporate the extensions into the workspace. For example, negotiation of the Basic Directory Contents package means that the service will return all attribute values that are described by that package in the formats detailed in Chapter 7.

After a workspace with the required features has been negotiated in this way, the application can use the workspace as required. It can create and manipulate OM objects using the OM functions, and can start one or more directory sessions using *Bind*( ).

Eventually, when it has completed its tasks, terminated all its directory sessions using *Unbind*( ), and released all its OM objects using *om_delete*( ), the application should ensure that resources associated with the workspace are freed by calling *Shutdown*( ). This will have the effect of deleting any private objects or service generated public objects that are in the workspace.

It is possible for an application to create more than one workspace using *Initialize*( ). This allows several uncoordinated activities, or libraries, to use these services without interfering with each other. Each such workspace is distinct.

3.3 Session

A session identifies to which directory user agent (DUA), and to which suite of directory system agents (DSAs), a particular directory operation will be sent. It contains some **Directory-BindArgument**s, such as the distinguished name of the requestor. The session is passed as the first argument to most interface functions.

A session is described by an OM object of OM class **Session**. It is created, and appropriate parameter values may be set using the object management functions. A directory session is then started with *Bind*( ) and later is terminated with *Unbind*( ). A session with default parameters can be started by passing the constant **Default-Session** (DS_DEFAULT_SESSION) as the **Session** argument to *Bind*( ).

Bind( ) must be called before the **Session** can be used as an argument to any other function in this interface. After *Unbind*( ) has been called, *Bind*( ) must be called again if another session is to be started.

The interface supports multiple concurrent sessions so that an application implemented as a single process, such as a server in a client-server model, can interact with the directory using several identities, and also so that a process can interact directly and concurrently with different parts of the directory.

Detailed specifications of the OM class **Session** are given in Chapter 5.

3.4 Context

The context defines the characteristics of the directory interaction that are specific to a particular directory operation, but often are used unchanged for many operations. Since these parameters are presumed to be relatively static for a given directory user during a particular directory interaction, these arguments are collected into an OM object, of OM class **Context**, which is supplied as the second argument of each directory service request. This serves to reduce the number of arguments passed to each function.

The context includes many administrative details, such as the **CommonArguments** defined in the Abstract Service, which affect the processing of each directory operation. These include a number of **ServiceControls**, which allow control over some aspects of the service. These include *options* such as **preferChaining**, **chainingProhibited**, **localScope**, **dontUseCopy** and **dontDereferenceAliases**, together with **priority**, **timeLimit**, **sizeLimit** and **scopeOfReferral**. Each of these is mapped onto an OM attribute in the context, and they are detailed in Chapter 5.

The effect is as if they were passed as a group of additional arguments on every function call. The value of each component of the context is determined when the interface function is called, and remains fixed throughout the operation.

All OM attributes in the **Context** have default values, some of which are locally administered. The constant **Default-Context** (DS_DEFAULT_CONTEXT) can be passed as the value of the **Context** argument to the interface functions, and has the same effect as a context OM object created with default values. The context must be a private object, unless it is **Default-Context**.

Detailed specifications of the OM class **Context** are given in Chapter 5.

3.5 Function Arguments

The Abstract Service defines specific arguments for each operation. These are mapped onto corresponding arguments to each interface function (which are also called input parameters). Although each service has different arguments, some specific arguments recur in several operations and these are briefly introduced here. Full details of these and all the other arguments are given in the function definitions of Chapter 4, and the OM class definitions of Chapter 5.

All arguments that are OM objects can generally be supplied to the interface functions as public objects (that is, descriptor lists) or as private objects. Private objects must be created in a workspace that was returned by *Initialize*( ). In some cases, constants can be supplied instead of OM objects.

Note that wherever a function is stated to accept an instance of a particular OM class as the value of an argument, it will also accept an instance of any subclass of the OM class. For example, most functions have a **Name** argument, which accepts values of OM class *Name*. It is always valid to supply an instance of the subclass **DS-DN** as the value of the argument.

3.5.1 Attribute and AVA

Each directory attribute is represented in the interface by an OM object of OM class **Attribute**. The type of the directory attribute is represented by an OM attribute, **Attribute-Type**, within the OM object. The values of the directory attribute are expressed as the values of the OM attribute **Attribute-Values**.

The representation of the attribute value depends on the attribute type and is determined as set out below. This lists the way in which an application program must supply values to the interface (for example, in the *Changes* argument to the *Modify-Entry*( ) function). The interface follows the same rules when returning attribute values to the application (for example, in the result of the *Read*( ) function).

- The first possibility is that the attribute type and the representation of the corresponding values may be defined in a package, such as the selected attribute types from the standards that are defined in the package in Chapter 7. In this case, if use of the package has been negotiated by the application using the *Version*( ) function, attribute values are represented as specified in that package, otherwise the next rule is followed. Additional directory attribute types and their OM representations may be defined by future versions of this specification or by vendor extensions.
- The second possibility is that the attribute type is not known and the value can be represented as an OM syntax corresponding to an ASN.1 simple type. In this case, the representation is the corresponding OM syntax.
- The final possibility is that the attribute type is not known and the value cannot be represented as an OM syntax corresponding to an ASN.1 simple type. In this case, the value is represented as an object of OM class *Encoding*.

Where attribute values have OM syntax String(*), they may be long, segmented strings and the functions *OM-Read*( ) and *OM-Write*( ) should be used to access them.

An attribute-value-assertion (AVA) is an assertion about the value of an attribute of an entry, and can be true, false or undefined. It consists of an attribute type and a single value. Loosely, the AVA is true if one of the values of the given attribute in the entry matches the given value.

An AVA is represented in the interface by an instance of OM class **AVA**, which is a subclass of **Attribute**, constrained to have exactly one value.

Information used by *Add-Entry*( ) to construct a new directory entry is represented by an OM object of OM class **Attribute-List**, which contains a single, multi-valued OM attribute whose values are OM objects of OM class **Attribute**.

3.5.2 Entry Information Selection

The **Selection** argument of the *Read*( ) and *Search*( ) operations tailors their results to obtain just part of the required entry. Information about all attributes, no attributes, or a named set, can be chosen. Attribute types are always returned, but the attribute values need not be.

The value of the argument is an instance of OM class **Entry-Information-Selection**, but one of the constants below can be used in simple cases.

- To verify the existence of an entry for the purported name, use the constant **Select-No-Attributes** (DS_SELECT_NO_ATTRIBUTES).
- To return just the types of all attributes, use the constant **Select-All-Types** (DS_SELECT_ALL_TYPES).
- To return the types and values of all attributes, use the constant **Select-All-Types-And-Values** (DS_SELECT_ALL_TYPES_AND_VALUES).

To choose a particular set of attributes, create a new instance of the OM class **Entry-Information-Selection** and set the appropriate OM attribute values using the OM functions.

3.5.3 Name

Most operations take a *Name* argument to specify the target of the operation. The name is represented by an instance of one of the subclasses of the OM class *Name*. This specification defines the subclass **DS-DN** to represent distinguished names and other names.

For directory interrogations, any aliases in the name will be dereferenced unless prohibited by the **Dont-Dereference-Aliases** service control. But for directory modifications, any aliases in the name will *not* be dereferenced.

RDNs are represented by an instance of one of the subclasses of the OM class *Relative-Name*. This specification defines the subclass **DS-RDN** to represent RDNs.

3.6 Function Results

All functions return a **Status** (which is the C function result), most return an **Invoke-ID**, which identifies the particular invocation, and the interrogation operations each return a **Result**. (The **Invoke-ID** and **Result** are returned using pointers that are supplied as arguments of the C function). These three kinds of function results are introduced below.

All OM objects returned by interface functions (results and errors) will be private objects in the workspace associated with the directory session in the call to *Bind*( ).

3.6.1 Invoke-ID

All interface functions that invoke a directory operation return an **Invoke-ID**- an integer that identifies the particular invocation of an operation. The **Invoke-ID** is only relevant for asynchronous operations and may be used later to receive the result and status, or to abandon them. Asynchronous operations are described in Section 3.7 on page 23, and the interface functions that can be used to start them are indicated by an *a* in Table 3-1 on page 17.

The numerical value returned from a call that successfully invokes an asynchronous operation is guaranteed to be unique amongst all outstanding operations in a given session.

The value returned for a synchronous operation is unspecified, as is that for a call that fails to invoke an operation.

3.6.2 Result

Directory interrogation operations return a result only if they succeed. All errors from these operations, including DAP errors, are reported in **Status**, described below, as are errors from all other operations.

The value returned by a function call that invokes an asynchronous operation is unspecified, as is that for a call that fails to invoke an operation. The result of an asynchronous operation is returned by a later call to *Receive-Result*( ).

The result of an interrogation is returned in a private object whose OM class is appropriate to the particular operation. The format of directory operation results is driven both by the Abstract Service and by the need to provide asynchronous execution of functions. To simplify processing of asynchronous results, the result of a single operation is returned in a single OM object (corresponding to the abstract result defined in the Standards). The components of the result of an operation are represented by OM attributes in the operation's result object. All information contained in the Abstract Service result is made available to the application program. The result is inspected using the functions provided in the referenced **XOM** Specification.

Only the interrogation operations produce results, and each type of interrogation has a specific OM class of OM object for its result. These OM classes, **Compare-Result**, **List-Result**, **Read-Result** and **Search-result**, are defined in Chapter 5. The results of the different operations share several common components, including the **CommonResults** defined in the Standards (see Referenced Documents), by inheriting OM attributes from the superclass *Common-Results*.

The actual OM class of the result can always be a subclass of that named, in order to allow flexibility for extensions. Thus, the function *OM-Instance*( ) should always be used when testing the OM class.

Any attribute values in the result are represented as discussed in Section 3.5.1 on page 20.

3.6.3 Status

Every interface function returns a *Status* value, which is either a constant or an object.

If it is a constant, then it is one of the following:

- [DS_SUCCESS]
- [DS_NO_WORKSPACE].

If it is an object, then it is one of the following:

- an attribute error (represented by a private object of class **Attribute-Error**)

- a referral (represented by a private object of class **Referral**)
- an error represented by a private object of one of the subclasses of class *Error*.

Other results of functions are not valid unless the status result has the value [DS_SUCCESS].

3.7 Synchronous and Asynchronous Operations

Asynchronous operations may be provided by the interface, determined for each operation by the value of the **Asynchronous** OM attribute in the **Context** passed to the interface function. The default value of this OM attribute is **false**, causing all operations to be synchronous. Asynchronous operation is an optional feature; operations are always synchronous on implementations that do not provide it. The value of **max-outstanding-operations**, described below, indicates the presence of the feature.

In synchronous mode, all functions wait until the operation is complete before returning. Thus the thread of control is blocked within the interface after calling a function, and it can make use of the result immediately after the function returns.

Note: In a multithreaded system, only one thread in the process is blocked, and use of asynchronous mode is likely to be rare on such systems. On conventional single-thread-per-process systems, the entire process is blocked, leading to the need for asynchronous mode).

In asynchronous mode, some functions return before the operation is complete. The application is then able to continue with other processing whilst the operation is being executed by the directory, and can then access the result by calling *Receive-Result*( ). An application may initiate several concurrent asynchronous operations on the same session before receiving any of the results, subject to the limit described below. The functions that can execute asynchronously are indicated in Table 3-1 on page 17.

An asynchronous function call returns an **Invoke-ID** of the operation to the application. The same **Invoke-id** can be passed to *Abandon*( ) to cause the operation to be abandoned, or to *Receive-Result*( ) to receive the results for that operation.

Implementations of the interface are free to return from asynchronous function calls as soon as possible, or may wait until the operation has been submitted to the underlying directory service. The actual policy used is implementation-defined.

Implementations also define a limit on the number of asynchronous operations that may be outstanding at any one time on any one session. An asynchronous operation is outstanding from the time that the function is called until the result is returned by *Receive-Result*( ) or abandoned by *Abandon*( ). The limit is given by the constant **max-outstanding-operations** (DS_MAX_OUTSTANDING_OPERATIONS), and it has the value zero if asynchronous operation is not supported. If the feature is present, it is guaranteed to be at least one, so an application can always use the interface in asynchronous operating mode. While this number of operations is outstanding, attempts to invoke further asynchronous operations will report a DS *Library-Error* (too-many-operations).

A synchronous call (other than *Abandon*( ) or *Receive-Result*( )) may return a *Library-Error* (mixed-synchronous), if it is made on a session on which there are any outstanding asynchronous operations. All asynchronous operations should be allowed to terminate, and their results should be obtained, before making a synchronous call on the same session.

If an error is detected before an asynchronous request is submitted to the directory, the function returns immediately and there is no outstanding operation generated. Other errors are notified

later by *Receive-Result*( ), when the result of the outstanding asynchronous operation is returned. All errors occurring during a synchronous request are reported when the function returns.

Details of error handling are given in Chapter 6.

Where vendors provide suitable system primitives (such as System V *poll*( ), or BSD (Berkeley Source Distribution) *select*( )), applications can obtain a file descriptor from the **Session** by inspecting the value of the OM attribute *File-Descriptor*. Applications may use the file descriptor to suspend the process until data is received on the particular file descriptor.

Applications should ensure that there are no outstanding asynchronous operations on a session when *Unbind*( ) is called on that session. Once *Unbind*( ) has been called, there is no way to determine whether any outstanding operations succeed or even whether they were ever sent to the directory. No errors or results of any kind are reported to the application. It is strongly recommended that *Receive-Result*( ) is called repeatedly until **Completion-Flag** takes the value **no-outstanding-operation**.

3.8 Security

It is not the purpose of this interface specification to constrain the security policy of any implementation or local administration. Such policies may differ widely according to the requirements of different user groups.

The standards (see the referenced **ISO 9594** documents) define several security features to protect the provision of the Directory Service. Defining a security interface in this specification could have the effect of constraining local security policy. Consequently, this specification does not define a security interface.

Security can be provided in several ways. One way is to provide the security in the standards below the interface by some private means. Alternatively, implementations may provide security by means of extensions to the interface. For example, some implementations may add extra OM attributes to OM classes such as **Context** and **Session**. A third way is to provide a separate security interface.

The standards also allow the directory to provide a convenient repository for authentication information. This authentication information is represented as OM classes that are defined in the *Strong Authentication Package* , which is defined in Chapter 7.

3.9 Other Features of the Interface

3.9.1 Automatic Connection Management

Implementations may provide automatic management of the association, or connection, between the user and the directory, making and releasing connections at its discretion. Such management is intended to bring benefits such as reduced communication charges. In order to allow this flexibility to the implementation, the interface does not specify when communication with a DSA takes place. In particular, it does not require that the **DirectoryBind** operation specified in the standards is performed when the *Bind*( ) interface function is called.

3.9.2 Automatic Continuation and Referral Handling

The interface provides automatic handling of continuation references and referrals in order to reduce the burden on application programs. These facilities can be inhibited to meet special needs.

A continuation reference describes how the performance of all or part of an operation can be continued at a different DSA or DSAs. A single continuation reference, returned as the entire response to an operation, is called a *Referral* and is classified as an error. One or more continuation references may also be returned as part of a **Partial-Outcome-Qualifier** returned from a *List*( ) or *Search*( ) operation.

A DSA will return a referral if it has administrative, operational or technical reasons for preferring not to chain. It may return a referral if **Chaining-Prohibited** is set in the **Context**, though it may instead report a Service-Error (**chaining-required**) in this circumstance.

By default, the implementation will use any continuation references it receives to try to contact the other DSA(s), and so progress the operation, whenever practical. It will only return the result or an error to the application after it has made this attempt. Note that continuation references may still be returned to the application, for example, if the relevant DSA cannot be contacted.

The default behaviour is the simplest for most applications, but if necessary the application can cause all continuation references to be returned to it. It does this by setting the value of the OM attribute *Automatic-Continuations* in the **Context** to **false**.

Chapter 4

Interface Functions

NAME

Abandon - abandon the result of a pending, asynchronously-executing operation

SYNOPSIS

```
#include <xds.h>

DS_status ds_abandon(
     OM_private_object    session,
     OM_sint              invoke_id
);
```

DESCRIPTION

This function abandons the result of an outstanding asynchronous function call. The function is no longer outstanding after this function returns, and the result will never be returned by *Receive-Result*( ).

Note that this function may, but need not, cause an **Abandon** operation to be requested of the directory. Such an **Abandon** operation may, but need not, cause the directory to abandon the outstanding asynchronous operation itself.

ARGUMENTS

Session (Object (Session))

The directory session in which the operation was submitted to the directory. This must be a private object.

Invoke-ID (Integer)

Selects the specific outstanding asynchronous operation submitted via the **Session** to be terminated. The operation must be an interrogation (that is, *Compare*( ), *List*( ), *Read*( ) or *Search*( )). The value of **Invoke-ID** must be that which was returned by the function call that initiated the asynchronous directory operation to be abandoned.

RESULTS

Status (Status)

Whether the function succeeded or not.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors:** bad-session, miscellaneous.

This function can return a **Communications-Error** or the following **Directory-error**: Abandon-Failed. Note that the result of the asynchronous operation will not be returned even if an Abandon-Failed error is returned.

This function can return the error constant [DS_NO_WORKSPACE].

NAME

Add-Entry - add a leaf entry to the directory information tree

SYNOPSIS

```
#include <xds.h>

DS_status ds_add_entry(
     OM_private_object   session,
     OM_private_object   context,
     OM_object           name,
     OM_object           entry,
     OM_sint            *invoke_id_return
);
```

DESCRIPTION

Adds a leaf entry to the directory. The entry may be either an object or an alias. The entry is only added if it conforms to the directory schema.

ARGUMENTS

Session (Object (Session))
The directory session through which to submit the request. This must be a private object.

Context (Object (Context))
The directory user context to use for this operation. Note that **size-Limit** and **dont-Dereference-Aliases** do not apply to this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT) .

Name (Object (Name))
The name of the entry to be added. The immediate superior of the new entry is determined by removing the last RDN component (which belongs to the new entry). The immediate superior should exist in the same DSA, otherwise the function may fail with an Update-Error (affects-multiple-DSAs). Any aliases in the name will *not* be dereferenced.

Entry (Object (Attribute-List))
The attribute information which, together with that from the RDN, constitutes the entry to be created. Note that an instance of OM class **Entry-Information** can be supplied as the value of this argument, since **Entry-Information** is a subclass of **Attribute-List**, and so can be used as the value of this argument.

RESULTS

Invoke-ID (Integer)
The **Invoke-ID** of an asynchronous directory operation.

Status (Status)
Whether the entry was added or not, if used synchronously, or whether the operation was initiated, if used asynchronously.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-name, bad-session, bad-workspace, miscellaneous, missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Attribute-Error, Name-Error, Referral, Security-Error, Service-Error, Update-Error.

The Update-Error (affects-multiple-DSAs), which is referred to in the argument descriptions, need not be returned if there is local agreement between the DSAs to allow the entry to be added.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

NAME

Bind - open a session with the directory

SYNOPSIS

```
#include <xds.h>

DS_status ds_bind(
     OM_object             session,
     OM_workspace          workspace,
     OM_private_object    *bound_session_return
);
```

DESCRIPTION

In order to allow automatic connection management, *Bind*( ) may or may not communicate with a DSA when it is called.

ARGUMENTS

Session (Object (Session))

Specifies a particular directory service provider, together with other details of the service required. This argument may be either a public object or a private object. The constant **Default-Session** (DS_DEFAULT_SESSION) may also be used as the value of this argument, causing a new session to be created with default values for all its OM attributes.

Workspace (Workspace)

Specifies the workspace (obtained from a call to *Initialize*( )) which is to be associated with the session. All function results from directory operations using this session will be returned as private objects in this workspace. If the **Session** argument is a private object, it must be a private object in this workspace.

RESULTS

Status (Status)

Whether or not the function completed successfully.

Bound-Session (Object (Session))

Upon successful completion, contains an instance of a directory session that may be used as an argument to other functions (for example, *Read*( )). This will be a new private object if the value of **Session** was **Default-Session** or a public object, otherwise it will be that supplied as an argument. The function will supply default values for any of the OM attributes that were not present in the **Session** instance supplied as an argument. It will also set the value of the **File-Descriptor** OM attribute (the value will be **No-Valid-File-Descriptor** (DS_NO_VALID-FILE-DESCRIPTOR) if the functionality is not supported).

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**: bad-session, miscellaneous, too-many-sessions.

The following **Directory-errors** may be returned by the function: Security-Error, Service-Error.

This function can return a **Communications-Error**.

This function can return the error constant [DS_INVALID_WORKSPACE].

SEE ALSO

Unbind( ) .

NAME

Compare - compare a purported attribute value with the attribute value stored in the directory for a particular entry

SYNOPSIS

```
#include <xds.h>

DS_status ds_compare(
     OM_private_object     session,
     OM_private_object     context,
     OM_object             name,
     OM_object             ava,
     OM_private_object    *result_return,
     OM_sint              *invoke_id_return
);
```

DESCRIPTION

Compares the value supplied in the given AVA with the value(s) of the same attribute type in the named entry.

The result of this operation can be abandoned.

ARGUMENTS

Session (Object (Session))
The directory session against which this operation is performed. This must be a private object.

Context (Object (Context))
The directory context to be used for this operation. Note that **size-Limit** does not apply to this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT) .

Name (Object (Name))
The name of the target object entry. Any aliases in the name will be dereferenced unless prohibited by the relevant service control.

AVA (Object (AVA))
The Attribute Value Assertion which specifies the attribute type and value to be compared with that in the entry.

RESULTS

Status (Status)
Whether the comparison was completed or not, if used synchronously, or whether the operation was initiated, if used asynchronously. Note that the operation fails and an error is returned either if the target object is not found or if it does not have an attribute of the required type.

Result (Object (Compare-Result))
Upon successful completion of a synchronous call, the result contains flags indicating whether the values matched and whether the comparison was made against the original entry. It also contains the distinguished name of the target object if an alias was dereferenced.

Invoke-ID (Integer)
The **Invoke-ID** of an asynchronous directory operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Error**s: bad-argument, bad-class, bad-context, bad-name, bad-session, bad-workspace, miscellaneous, missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Attribute-Error, Name-Error, Referral, Security-Error, Service-Error.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

SEE ALSO

Abandon( ).

NAME

Initialize - initialise the interface and allocate a workspace

SYNOPSIS

```
#include <xds.h>

OM_workspace ds_initialize(
     void
);
```

DESCRIPTION

This function performs any necessary initialisation of the interface and allocates a workspace. It must be called before any other directory interface functions are called. It may be called multiple times, in which case each call returns a workspace which is distinct from other workspaces created by *Initialize*( ) but not yet deleted by *Shutdown*( ).

ARGUMENTS

None.

RESULTS

Workspace (Workspace)

Upon successful completion, contains a handle to a workspace in which OM objects can be created and manipulated. Objects created in this workspace, and only such objects, may be used as arguments to the other directory interface functions. This function returns NULL if it fails.

ERRORS

None.

SEE ALSO

Shutdown( ).

NAME

List - enumerate the immediate subordinates of a particular directory entry

SYNOPSIS

```
#include <xds.h>

     DS_status ds_list(
     OM_private_object   session,
     OM_private_object   context,
     OM_object           name,
     OM_private_object  *result_return,
     OM_sint            *invoke_id_return
);
```

DESCRIPTION

This function is used to obtain a list of the immediate subordinates of the named entry. The list may be incomplete in some circumstances.

The result of this operation can be abandoned.

ARGUMENTS

Session (Object (Session))
The directory session against which this operation is performed. This must be a private object.

Context (Object (Context))
The directory context to be used for this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT) .

Name (Object (Name))
The name of the object entry whose immediate subordinates are to be listed. Any aliases in the name will be dereferenced unless prohibited by the relevant service control.

RESULTS

Status (Status)
Takes the value **success** if the named object was located (even if there are no subordinates) and takes an error value if not, when the subroutine is used synchronously. Reports whether the operation was initiated, if used asynchronously.

Result (Object (List-Result))
Upon successful completion of a synchronous call, the result contains some information about the target object's immediate subordinates. It also contains the distinguished name of the target object, if an alias was dereferenced to find it. Aliases in the subordinate names are not dereferenced. Additionally there may be a partial outcome qualifier, which indicates that the result is incomplete. It also explains why (for example, because the time limit expired) and contains information that may be helpful when attempting to complete it.

Invoke-ID (Integer)
The **Invoke-ID** of an asynchronous directory operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors:** bad-argument, bad-class, bad-context, bad-name, bad-session, bad-workspace, miscellaneous, missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Name-Error, Referral, Security-Error, Service-Error.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

SEE ALSO

Abandon( ).

NAME

Modify-Entry - perform an atomic modification on a directory entry

SYNOPSIS

```
#include <xds.h>

DS_status ds_modify_entry(
     OM_private_object    session,
     OM_private_object    context,
     OM_object            name,
     OM_object            changes,
     OM_sint             *invoke_id_return
);
```

DESCRIPTION

This function is used to make a series of one or more of the following changes to a single directory entry:

- add a new attribute (**add-attribute**)
- remove an attribute (**remove-attribute**)
- add attribute values (**add-values**)
- remove attribute values (**remove-values**).

Values may be replaced by a combination of adding values and removing values in a single operation. The RDN can only be changed by using the *Modify-RDN*( ) function.

The result of the operation is as if each modification is made in the order specified in the *Changes* argument. If any of the individual modifications fails, then an Attribute-Error is reported and the entry is left as it was prior to the whole operation. The operation is atomic; either all or none of the changes are made. The directory checks that the resulting entry conforms to the directory schema.

ARGUMENTS

Session (Object (Session))
: The directory session against which this operation is performed. This must be a private object.

Context (Object (Context))
: The directory context to be used for this operation. Note that **size-Limit** and **dont-Dereference-Aliases** do not apply to this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT) .

Name (Object (Name))
: The name of the target object entry. Any aliases in the name will *not* be dereferenced.

Changes (Object (Entry-Mod-List))
: A sequence of modifications to the named entry.

RESULTS

Status (Status)
Takes the value **success** if all the modifications succeeded and takes an error value if not, when the function is used synchronously. Reports whether the operation was initiated, if used asynchronously.

Invoke-ID (Integer)
The **Invoke-ID** of an asynchronous directory operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors:** bad-argument, bad-class, bad-context, bad-name, bad-session, bad-workspace, miscellaneous, missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Attribute-Error, Name-Error, Referral, Security-Error, Service-Error, Update-Error.

Attempting to use **add-attribute** to add an existing attribute results in an Attribute-Error. Attempting to use **add-values** to add an existing value results in an Attribute-Error, as does an attempt to add a value to a non-existent attribute type. Attempting to use **remove-attribute** to remove a non-existing attribute results in an Attribute-Error, while an attempt to remove an attribute that is part of the object's RDN results in an Update-Error. Attempting to use **remove-values** to remove a non-existing value results in an Attribute-Error, while an attempt to remove a value of an attribute that is part of the object's RDN, or to modify the object class attribute, results in an Update-Error.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

NAME

Modify-RDN - change the relative distinguished name (RDN) of a leaf entry

SYNOPSIS

```
#include <xds.h>

DS_status ds_modify_rdn(
     OM_private_object    session,
     OM_private_object    context,
     OM_object            name,
     OM_object            new_RDN,
     OM_boolean           delete_old_RDN,
     OM_sint             *invoke_id_return
);
```

DESCRIPTION

This function is used to change the RDN of a leaf entry (either an object entry or an alias entry).

ARGUMENTS

Session (Object (Session))

The directory session against which this operation is performed. This must be a private object.

Context (Object (Context))

The directory context to be used for this operation. Note that **size-Limit** and **dont-Dereference-Aliases** do not apply to this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT) .

Name (Object (Name))

The current name of the target leaf entry. Any aliases in the name will *not* be dereferenced. The immediate superior should not have any non-specific subordinate references, otherwise the function may fail with an Update-Error (affects-multiple-DSAs). (A non-specific subordinate reference is an indication that another DSA holds some number of children, but does not indicate their RDNs. This means that it is not possible to check the uniqueness of the requested new RDN within a single DSA.)

New-RDN (Object (Relative Name))

The requested new Relative Distinguished Name.

If an attribute value in the new RDN does not already exist in the entry (either as part of the old RDN or as a non-distinguished value), the new value is added. If it cannot be added, an error is reported.

Delete-Old-RDN (Boolean)

If this value is **true**, all attribute values that are in the old RDN but not in the new RDN are deleted. If the value is **false**, the old values should remain in the entry (not as part of the RDN). The value must be **true** when a single value attribute in the RDN has its value changed by the operation. If this operation removes the last attribute value of an attribute, that attribute will be deleted.

RESULTS

Status (Status)

Whether the name of the entry was changed or not, if used synchronously, or whether the operation was initiated, if used asynchronously.

Invoke-ID (Integer)
The **Invoke-ID** of an asynchronous directory operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-name, bad-session, bad-workspace, miscellaneous, missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Attribute-Error, Name-Error, Referral, Security-Error, Service-Error, Update-Error.

The Update-Error (affects-multiple-DSAs), which is referred to in the argument descriptions, need not be returned if there is local agreement between the DSAs to allow the entry to be modified.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

NAME

Read - query information on an entry by name

SYNOPSIS

```
#include <xds.h>

DS_status ds_read(
     OM_private_object    session,
     OM_private_object    context,
     OM_object            name,
     OM_object            selection,
     OM_private_object   *result_return,
     OM_sint             *invoke_id_return
);
```

DESCRIPTION

Read is used to extract information from an explicitly named entry. It can also be used to verify a distinguished name.

The result of this operation can be abandoned.

ARGUMENTS

Session (Object (Session))
The directory session against which this operation is performed. This must be a private object.

Context (Object (Context))
The directory context to be used for this operation. Note that **size-Limit** does not apply to this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT).

Name (Object (Name))
The name of the target object entry. Any aliases in the name will be dereferenced unless prohibited by the **Dont-Dereference-Aliases** service control.

Selection (Object (Entry-Information-Selection))
Specifies what information from the entry is requested. Information about no attributes, all attributes or just for a named set can be chosen. Attribute types are always returned, but the attribute values need not be. The possible values of this argument are set out in

RESULTS

Status (Status)
Whether the read was completed or not, if used synchronously, or whether the operation was initiated, if used asynchronously.

Result (Object (Read-Result))
Upon successful completion of a synchronous call, the result contains the distinguished name of the target object, and a flag indicating whether the result came from the original entry or a copy, as well as any requested attribute types and values. Attribute information is only returned if access rights are sufficient.

Invoke-ID (Integer)
The **Invoke-ID** of an asynchronous directory operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**: bad-argument, bad-attribute, bad-class, bad-context, bad-name, bad-session, bad-workspace, miscellaneous,

missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Attribute-Error, Name-Error, Referral, Security-Error, Service-Error.

An Attribute-Error (no-such-attribute) is reported if an explicit list of attributes is specified by the *Selection* argument, but none of them are present in the entry. This error is not reported if any of the selected attributes are present.

A Security-Error (insufficient-access-rights) will only be reported where access rights preclude the reading of all requested attribute values.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

SEE ALSO

Abandon( ).

NAME

Receive-Result - retrieve the result of an asynchronously executed operation

SYNOPSIS

```
#include <xds.h>

DS_status ds_receive_result(
     OM_private_object     session,
     OM_sint               invoke_id,
     OM_uint              *completion_flag_return,
     DS_status            *operation_status_return,
     OM_private_object    *result_return,
     OM_sint              *invoke_id_return
);
```

DESCRIPTION

This function is used to retrieve the completed result of a previous asynchronous operation.

The function results include two status indications. One, called **Status**, indicates that this function call itself was successful; it is always returned. The other, called **Operation-Status**, is used to return the status of the completed asynchronous operation, and is only returned if there is one.

ARGUMENTS

Session (Object (Session))

The directory session against which this operation is performed. This must be a private object.

Invoke-Id (Integer)

The invocation identifier of the asynchronous directory operation whose result is to be returned.

If the value of this argument is **any-operation** (DS_ANY_OPERATION), then the service will return the result of any asynchronous directory operation that has completed. The service will prioritise the retrieval of the completed directory operations in an implementation defined manner.

RESULTS

Status (Status)

Takes an error value if one of the errors listed below occurred during the execution of this function, and **success** otherwise.

Completion-Flag (Unsigned-Integer)

One of the following values to indicate the status of the outstanding asynchronous directory operation(s) specified by the invoke_id input argument:

- **completed-operation** [DS_COMPLETED_OPERATION]. The specified directory operation has completed and its result is made available or, if **any-operation** was specified, at least one outstanding directory operation has completed and its result is made available.
- **outstanding-operations** [DS_OUTSTANDING_OPERATIONS]. There are outstanding asynchronous directory operations but none has yet completed.
- **no-outstanding-operation** [DS_NO_OUTSTANDING_OPERATION]. There are no outstanding asynchronous directory operations.

- **other-completed-operations** [DS_OTHER_COMPLETED_OPERATIONS]. A particular directory operation was specified; that directory operation has not completed, but one or more other directory operations have completed.

This result is valid if the Status has the value **success.**

Upon successful return with **completion_flag** having the value **completed-operation**, the status and invocation identifier of the completed directory operation are returned.

Operation-Status (Status)
Takes an error value if an error occurred during the execution of the asynchronous directory operation, and **success** otherwise. The possible error values are listed for each individual operation in the corresponding function description.

This result is only valid if the **Status** has the value **success** and **Completion-Flag** has the value **completed-operation**.

Result (Object *)
The result of the completed asynchronous operation. Its value will be the constant **Null-Result** [DS_NULL_RESULT] if the operation was one that does not return a result (that is, *Add-Entry*( ), *Modify-Entry*( ), *Modify-RDN*( ) or *Remove-Entry*( )). Otherwise, the OM object's OM class is that of the result of the asynchronous operation, and can be determined by using the OM functions.

This result is only valid if the **Status** has the value **success, Completion-Flag** has the value **completed-operation** and **Operation-Status** has the value **success.**

Invoke-ID (Integer)
The **Invoke-ID** of the operation whose result is being returned.

This result is valid if the **Status** has the value **success** and **Completion-Flag** has the value **completed-operation**.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**: bad-session, bad-workspace, miscellaneous.

This function does not report any **Directory-errors**, or a **Communications-Error**, in its **Status** result. (Any such errors related to the completed asynchronous operation are reported in **Operation-Status**, as described above).

This function can return the error constant [DS_NO_WORKSPACE].

NAME

Remove-Entry - remove a leaf entry from the directory information tree

SYNOPSIS

```
#include <xds.h>

DS_status ds_remove_entry(
OM_private_object      session,
OM_private_object      context,
OM_object              name,
OM_sint               *invoke_id_return
);
```

DESCRIPTION

This function is used to remove a leaf entry from the directory (either an object entry or an alias entry).

ARGUMENTS

Session (Object (Session))

The directory session against which this operation is performed. This must be a private object.

Context (Object (Context))

The directory context to be used for this operation. Note that **size-Limit** and **dont-Dereference-Aliases** do not apply to this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT).

Name (Object (Name))

The name of the target object entry. Any aliases in the name will *not* be dereferenced.

RESULTS

Status (Status)

Whether the entry was deleted or not, if used synchronously, or whether the operation was initiated, if used asynchronously.

Invoke-ID (Integer)

The **Invoke-ID** of an asynchronous directory operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Error**s: bad-argument, bad-context, bad-name, bad-session, bad-workspace, miscellaneous, missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Name-Error, Referral, Security-Error, Service-Error, Update-Error.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

NAME

Search - find entries of interest in a portion of the directory information tree

SYNOPSIS

```
#include <xds.h>

DS_status ds_search(
     OM_private_object    session,
     OM_private_object    context,
     OM_object            name,
     OM_sint              subset,
     OM_object            filter,
     OM_boolean           search_aliases,
     OM_object            selection,
     OM_private_object   *result_return,
     OM_sint             *invoke_id_return
);
```

DESCRIPTION

This function is used to search a portion of the directory and return selected information from entries of interest. The information may be incomplete in some circumstances.

The result of this operation can be abandoned.

ARGUMENTS

Session (Object (Session))
The directory session against which this operation is performed. This must be a private object.

Context (Object (Context))
The directory context to be used for this operation. This argument must be a private object or the constant **Default-Context** (DS_DEFAULT_CONTEXT) .

Name (Object (Name))
The name of the object entry that forms the base of the search. Any aliases in the name will be dereferenced unless prohibited by the **dont-Dereference-Aliases** service control.

Subset (Integer)
Specifies the portion of the directory information tree to be searched. Its value must be one of:

— **base-object** (DS_BASE_OBJECT)
Search just the given object entry.

— **one-level** (DS_ONE_LEVEL)
Search just the immediate subordinates of the given object entry.

— **whole-subtree** (DS_WHOLE_SUBTREE)
Search the given object and all its subordinates.

Filter (Object (Filter))
The filter is used to eliminate entries from the search that are not wanted. Information will only be returned on entries that satisfy the filter. The constant **No-Filter** (DS_NO_FILTER) may be used as the value of this argument if all entries should be searched, and none eliminated. This corresponds to a filter with a **Filter-Type** value of **and**, and no values of the OM attributes **Filters** or **Filter-Items**.

Search-Aliases (Boolean)
Any aliases in the subordinate entries being searched will be dereferenced if the value of this argument is **true**, and will not be dereferenced if its value is **false**.

Selection (Object (Entry-Information-Selection))
Specifies what information from the entry is requested. Information about no attributes, all attributes or just for a named set can be chosen. Attribute types are always returned, but the attribute values need not be. The possible values of this argument are set out in Section 3.5.2 on page 21.

RESULTS

Status (Status)
Takes the value **success** if the named object was located and takes an error value if not, when the function is used synchronously. Reports whether the operation was initiated, if used asynchronously.

Result (Object (Search-Result))
Upon successful completion of a synchronous call, the result contains the requested information from each object in the search space that satisfied the filter. The distinguished name of the target object is present if an alias was dereferenced. Additionally there may be a partial outcome qualifier, which indicates that the result is incomplete. It also explains why it is not complete and how it could be completed.

Invoke-ID (Integer)
The **Invoke-ID** of an asynchronous directory operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**: bad-argument, bad-class, bad-context, bad-name, bad-session, bad-workspace, miscellaneous, missing-type, too-many-operations.

The following **Directory-errors** may be returned by the function (or by *Receive-Result*( ) when used asynchronously): Attribute-Error, Name-Error, Referral, Security-Error, Service-Error.

Note that an unfiltered search of just the base object succeeds even if none of the requested attributes are found, whilst *Read*( ) fails with the same selected attributes.

A Security-Error (insufficient-access-rights) will only be reported where access rights preclude the reading of all requested attribute values.

This function can return a **Communications-Error**.

This function can return the error constant [DS_NO_WORKSPACE].

SEE ALSO

Abandon( ).

NAME

Shutdown - delete a directory workspace

SYNOPSIS

```
#include <xds.h>

DS_status ds_shutdown(
     OM_workspace     workspace
);
```

DESCRIPTION

This function deletes a workspace established by *initialize*( ). All service generated objects associated with the workspace are deleted, and thus are no longer accessible. This may enable the service to release resources.

All sessions associated with the workspace must be terminated by calling *unbind*( ) prior to calling *shutdown*( ). No other interface function must reference the workspace after it has been deleted.

ARGUMENTS

Workspace (Workspace)
Specifies the workspace (obtained from a call to *Initialize*( )) which is to be deleted.

RESULTS

None.

ERRORS

This function can return the error constant [DS_INVALID_WORKSPACE].

This function does not return a **Communications-Error** or any **Directory-errors.**

SEE ALSO

Initialize( ).

NAME

Unbind - unbind from a directory session

SYNOPSIS

```
#include <xds.h>

DS_status ds_unbind(
     OM_private_object    session
);
```

DESCRIPTION

This function terminates the given directory session, and makes the argument unavailable for use with other interface functions (except *Bind*( )).

Note that this means the results of any outstanding asynchronous operations that were initiated using the given **Session** can no longer be received, and it is not possible to know whether they succeeded. Any such operations may be carried out or may be terminated prematurely. For this reason it is recommended that all outstanding asynchronous operations are processed using *Receive-result*( ) before *Unbind*( ) is called.

The unbound session may be used again as an argument to *Bind*( ), possibly after modification by the object management functions. When it is no longer required, it must be deleted using the object management functions.

ARGUMENTS

Session (Object (Session))

The directory session that is to be unbound. This must be a private object. The value of the **File-Descriptor** OM attribute will be **No-Valid-File-Descriptor** (DS_NO_VALID_FILE_DESCRIPTOR) if the function succeeds. The other OM attributes will be unchanged.

RESULTS

Status (Status)

Takes the value **success** if **Session** was unbound, and takes an error value if not.

ERRORS

This function can return a **System-Error** or one of the following **Library-Error**s: bad-session, miscellaneous.

This function does not return a **Communications-Error** or any **Directory-errors**.

This function can return the error constant [DS_NO_WORKSPACE].

SEE ALSO

Bind( ).

NAME

Version - negotiate features of the interface and service

SYNOPSIS

```
#include <xds.h>

DS_status ds_version(
     DS_feature       feature_list[],
     OM_workspace     workspace
);
```

DESCRIPTION

This function negotiates features of the interface, which are represented by Object Identifiers, and associates them with a workspace. The Basic-Directory-Contents, Strong-Authentication, and MHS-Directory-User packages, specified in Chapter 7, are the only negotiable features in this specification. Features may also include vendor extensions and new features in future versions of this specification.

ARGUMENTS

Feature-List (Feature-List)

An ordered sequence of features, each represented by an object identifier. The sequence is terminated by an object identifier having no components (a length of zero and any value of the data pointer in the C representation).

Workspace (Workspace)

Specifies the workspace (obtained from a call to *Initialize*( )) for which the features are to be negotiated. The features will be in effect for operations which use the workspace or directory sessions associated with the workspace.

RESULTS

Status (Status)

Whether or not the function completed successfully.

Activated (Boolean-List)

If the function completed successfully, this result contains an ordered sequence of Boolean values, with the same number of elements as the **Feature-List**. If true, each value indicates that the corresponding feature is now part of the interface. If false, each value indicates that the corresponding feature is not available.

In the C binding, this result is combined with the **Feature-List** argument as a single array of structures of type *DS_feature*, which is defined as:

```
typedef struct
   {
       OM_object_identifier      feature,
       OM_boolean                activated,
   }
DS_feature;
```

ERRORS

This function can return a **System-Error** or the following **Library-Error**: miscellaneous.

This function does not return a **Communications-Error**, or any **Directory-errors**.

This function can return the error constant [DS_INVALID_WORKSPACE].

Chapter 5

Interface Class Definitions

Note: Throughout this document, care is taken to distinguish between OM classes and directory classes, and between OM attributes and directory attributes. (In both cases, the former is a construct of the closely associated object management interface, while the latter is a construct of the Directory Service to which the interface provides access.) The terms ''object class'' and ''attribute'' denote the directory constructs, while the phrases ''OM class'' and ''OM attribute'' denote the object management ones.

5.1 Introduction

This Chapter defines the OM classes in alphabetical order that constitute the Directory Service (DS) package. The errors defined in Chapter 6 also belong to this package. The Object-Identifier associated with this package is

{iso(1) member-body(2) us(840) IEEE-P1224.2(10014) dsp(0)}

(with the encoding *"\x2a\x86\x48\xce\x1e\x0"*). This Object-Identifier is represented by the constant **Service-Package** (DS_SERVICE_PACKAGE).

The concepts of object management were briefly described in Section 1.4 on page 5, and the notation is introduced below. Both are fully explained in the referenced **XOM** Specification.

Each OM class is described in a separate section, which identifies the OM attributes specific to that OM class. The OM classes and OM attributes for each OM class are listed in alphabetical order. The OM attributes that may be found in an instance of an OM class are those OM attributes specific to that OM class and those inherited from each of its superclasses. The OM class-specific OM attributes are defined in a table. The table gives the name of each OM attribute, the syntax of each of its values, any restrictions upon the length (in bits, octets (bytes), or characters) of each value, any restrictions upon the number of values, and the value, if any, the *OM_Create*( ) function supplies.

The constants that represent the OM classes and OM attributes in the C binding are defined in the **<xds.h>** header.

Vendor Extensions

Vendors may provide additional OM attributes in their implementation of particular OM classes, and their individual documentation will give details of the specification and usage of these. The presence of extensions can be negotiated by use of the *Version*( ) function.

All such OM attributes will have default values that lead to the behaviour described in this specification.

5.2 Class Hierarchy

This Section depicts the hierarchical organisation of the OM classes defined in this Chapter, and thus shows which OM classes inherit additional OM attributes from their superclasses. Subclassification is indicated by indentation, and the names of abstract OM classes are rendered in italics. Thus, for example, the concrete class **Presentation-Address** is an immediate subclass of the abstract class *Address*, which in turn is an immediate subclass of the abstract class *Object*.

Object (defined in the referenced **XOM** Specification
- **Access-Point**
- *Address*
  - **Presentation-Address**
- **Attribute**
  - **AVA**
  - **Entry-Mod**
  - **Filter-Item**
- **Attribute-List**
  - **Entry-enfo**
- *Common-Results*
  - **Compare-Result**
  - **List-Info**
  - **Read-Result**
  - **Search-Info**
- **Context**
- **Continuation-Reference**
- **Entry-Info-Selection**
- **Entry-Mod-List**
- *Error* (see Chapter 6)
- **Extension**
- **Filter**
- **List-Info-Item**
- **List-Result**
- *Name*
  - **DS-DN**
- **Operation-Progress**
- **Partial-Outcome-Qual**
- *Relative-Name*
  - **DS-RDN**
- **Search-Result**
- **Session**

The application is not permitted to create or modify instances of some OM classes, because these OM classes are only returned by the interface and never supplied to it. These OM classes are: **Access-Point**, **Compare-Result**, **Continuation-Reference**, all subclasses of *Error*, **List-Information**, **List-Information-Item**, **List-Result**, **Operation-Progress**, **Partial-Outcome-Qualifier**, **Read-Result**, **Search-Info**, **Search-Result**.

This specification does not mandate that any OM classes are encodable using *OM-Encode*( ) and *OM-Decode*( ).

5.3 Access-Point

An instance of OM class **Access-Point** identifies a particular point at which access to a DSA can occur.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Address	Object(Address)	-	1	-
AE-Title	Object(Name)	-	1	-

Table 5-1 OM Attributes of an Access-Point

Address
The address of the DSA, for use in communications to it.

AE-Title
The name of the DSA.

5.4 Address

The OM class *Address* represents the address of a particular entity or service (such as a DSA).

It is an abstract class that has the OM attributes of its superclass (*Object*) and no other OM attributes.

An address is an unambiguous name, label or number that identifies the location of the entity or service. All addresses are represented as instances of some subclass of this OM class. The only subclass defined in this specification is **Presentation-Address**, which is the presentation address of an OSI application entity, used for OSI communications with it. Vendors may define additional subclasses to represent other kinds of address.

5.5 Attribute

An instance of OM class **Attribute** is an attribute of an object and thus a component of its directory entry.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attribute-Type	String(Object-Identifier)	-	1	-
Attribute-Values	any	-	0 or more	-

Table 5-2 OM Attributes of an Attribute

Attribute-Type
The attribute type, which indicates the class of information given by this attribute.

Attribute-Values
The attribute values. The OM value syntax and the number of values allowed for this OM attribute are determined by the value of the **Attribute-Type** OM attribute in accordance with the rules set out in Section 3.5.1 on page 20.

Where the values of this OM attribute have syntax String(*), they may be long, segmented strings. For this reason, the functions *OM-Read*( ) and *OM-Write*( ) should be used to access all String(*) values.

Note that a directory attribute must always have at least one value, even though instances of this OM class are permitted to have none.

5.6 Attribute-List

An instance of OM class **Attribute-List** is a list of directory attributes.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attributes	Object(Attribute)	-	0 or more	-

Table 5-3 OM Attributes of an Attribute-List

Attributes
The attributes that will constitute a new object's directory entry, or those selected from an existing entry.

5.7 AVA

An instance of OM class **AVA** (Attribute Value Assertion) is a proposition concerning the values of a directory entry.

An instance of this OM class has the OM attributes of its superclasses (*Object*, **Attribute**) and no other OM attributes. There is an additional constraint on this OM class, in that there must be exactly one value of the OM attribute **Attribute-Values**. The **Attribute-Type** remains single-valued. The OM value syntax of **Attribute-Values** must conform to the rules set out in Section 3.5.1 on page 20.

5.8 Common-Results

The OM class *Common-Results* comprises results that are returned by, and thus common to, the directory interrogation operations.

It is an abstract OM class, which has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Alias-Dereferenced	Boolean	-	1	-
Performer	Object(Name)	-	0-1	-

Table 5-4 OM Attributes of a Common-Results

Alias-Dereferenced
Indicates whether the name of the target object that was passed as a function argument included an alias that was dereferenced to determine the distinguished name.

Performer
When present, gives the distinguished name of the performer of a particular operation. It may be present when the result is signed, and holds the name of the DSA that signed the result.

5.9 Compare-Result

An instance of OM class **Compare-Result** comprises the results of a successful call to the *Compare*( ) function.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Common-Results*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
From-Entry	Boolean	-	1	-
Matched	Boolean	-	1	-
Object-Name	Object(Name)	-	0-1	-

Table 5-5 OM Attributes of a Compare-Result

From-Entry
Indicates whether the assertion was tested against the specified object's entry, rather than a copy of the entry.

Matched
Indicates whether the assertion specified as an argument proved true. It takes the value **true** if the values were compared and matched, and **false** otherwise.

Object-Name
The distinguished name of the target object of the operation. It will be present if the OM attribute **Alias-Dereferenced**, inherited from the superclass *Common-Results*, is **true**.

5.10 Context

An instance of OM class **Context** comprises per-operation arguments that are accepted by most of the interface functions.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Common Arguments				
Aliased-RDNs	Integer	-	0-1	-
Ext	Object(Ext)	-	0 or more	-
Operation-Progress	Object(Operation-Progress)	-	1	Operation-Not-Started
Service Controls				
Chaining-Prohib	Boolean	-	1	*local*[1]
Dont-Dereference-Aliases	Boolean	-	1	false
Dont-Use-Copy	Boolean	-	1	*local*[1]
Local-Scope	Boolean	-	1	*local*[1]
Prefer-Chaining	Boolean	-	1	*local*[1]
Priority	Enum(Priority)	-	1	*local*[1]
Scope-Of-Referral	Enum(Scope-Of-Referral)	-	0-1	*local*[1]
Size-Limit	Integer	-	0-1	*local*[1]
Time-Limit	Integer	-	0-1	*local*[1]
Local Controls				
Asynchronous	Boolean	-	1	false
Automatic-Continuation	Boolean	-	1	true

[1]The default values of these OM attributes are locally-administered.

Table 5-6 OM Attributes of a Context

The context collects together several arguments passed to interface functions, which are presumed to be relatively static for a given directory user during a particular directory interaction. The context is passed as an argument to each function that interrogates or updates the directory. Although the presumption is that infrequent changes to the context will be made, the value of each argument can be changed between every operation if required. Each argument is represented by one of the OM attributes of the **Context** OM class.

The context contains the Common Arguments defined in the standards (see reference **ISO 9594**), except that all security information is omitted for reasons discussed in Section 3.8 on page 24. These are made up of a number of service controls, explained below, possible extensions in the *Extensions* OM attribute, and operation progress and alias dereferencing information in the *Operation-Progress* OM attribute. It also contains a number of arguments that provide local control over the interface.

The OM attributes of the **Context** OM class are described below.

Common Arguments

Aliased-RDNs
Indicates to the Directory that the object component of the operation argument was created by dereferencing of an alias on an earlier operation attempt. This value would have been set in the Referral response of the previous operation.

Extensions
Any future standardised extensions that should be applied to the directory operation.

Operation-Progress
The state that the directory service is to assume at the start of the operation. This OM attribute will normally take its default value, which is the value *Operation-Not-Started {DS_OPERATION_NOT_STARTED}* described in the **Operation-Progress** OM class definition.

Service Controls

Chaining-Prohibited
Indicates that chaining, and other methods of distributing the request around the directory, are prohibited.

Dont-Dereference-Aliases
Indicates that any alias used to identify the target entry of an operation is not to be dereferenced. This allows interrogation of alias entries (aliases are never dereferenced during updates).

Dont-Use-Copy
Indicates that the request is to be satisfied only by access to directory entries, and not by use of copies of entries. This includes both copies maintained in other DSAs by bilateral agreement, and locally cached copies.

Local-Scope
Indicates that the directory request is to be satisfied locally. The meaning of this option is configured by an administrator. (The option typically restricts the request to a single DSA or DMD.)

Prefer-Chaining
Indicates that chaining is preferred to referrals when necessary. The directory is not obliged to follow this preference, and may return a referral even if it is set.

Priority
The priority, relative to other directory requests, at which the directory is to attempt to satisfy the request. This is not a guaranteed service since there is no directory-wide queueing. Its value must be one of:

- **low**
- **medium**
- **high**.

Scope-Of-Referral
The portion of the directory to which referrals are to be limited. This includes Referral errors and Partial Outcome Qualifiers. Its value must be one of:

- **country**
  meaning DSAs within the country in which the request originates

- **DMD**
  meaning DSAs within the DMD in which the request originates.

Scope-of-Referral is an optional attribute. The lack of this attribute in a Context object indicates that the scope is not limited.

Size-Limit
If present, the maximum number of objects about which *List*( ) or *Search*( ) should return information. If this limit is exceeded, information is returned about exactly this number of objects. Which objects are chosen is unspecified (since it may depend on the timing of interactions between DSAs, among other reasons).

Time-Limit
If present, the maximum elapsed time, in seconds, within which the service should be provided (not the processing time devoted to the request). If this limit is reached, a Service-Error (time-limit-exceeded) is returned except for the *List*( ) or *Search*( ) operations, which return an arbitrary selection of the accumulated results.

Local Controls

Asynchronous
Indicates that the interface should operate asynchronously or not, as detailed in Section 3.7 on page 23. The value is one of:

- **false**
  meaning that the operation is to be performed sequentially (synchronously), with the application being blocked until a result or error is returned.
- **true**
  meaning that the operation is to be performed asynchronously (non-blocking). The application can perform multiple concurrent asynchronous operations and can associate a result obtained from *Receive-Result*( ) with the original operation. The maximum number of outstanding concurrent operations is implementation-defined, and is reflected by the value of the constant **max-outstanding-operations** (DS_MAX_OUTSTANDING_OPERATIONS).

Automatic-Continuation
Indicates the requestor's requirement for continuation reference handling, including referrals and those in partial outcome qualifiers. The value is one of:

- **false**
  meaning that the interface returns all continuation references to the application program.
- **true**
  meaning that continuation references are automatically processed and the subsequent results returned to the application instead of the continuation reference(s), whenever practical. This is much simpler, unless the application has special requirements. Note that continuation references may still be returned to the application, for example, if the relevant DSA cannot be contacted.

Applications can assume that an object of OM class **Context**, created with default values of all its OM attributes, will work with all the interface functions. Local administrators should ensure that this is the case. The constant **Default-Context** (DS_DEFAULT_CONTEXT) can be used as an argument to interface functions instead of creating an OM object with default values.

5.11 Continuation-Reference

An instance of OM class **Continuation-Reference** comprises the information that enables a partially completed directory request to be continued (for example, following a referral).

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Access-Points	Object(Access-Point)	-	1 or more	-
Aliased-RDNs	Integer	-	1	-
Operation-Progress	Object(Operation-Progress)	-	1	-
RDNs-Resolved	Integer	-	0-1	-
Target-Object	Object(Name)	-	1	-

Table 5-7 OM Attributes of a Continuation-Reference

Access-Points
The names and presentation addresses of the DSAs at all of which the directory request should be continued.

Aliased-RDNs
Indicates how many (if any) of the RDNs in the target name have been produced by dereferencing an alias. Its value is zero if no aliases have been dereferenced. This value should be used in the Context of any continued operation.

Operation-Progress
The state at which the directory request must be continued. This value should be used in the **Context** of any continued operation.

RDNs-Resolved
The number of RDNs, in the supplied object name, that have been resolved (using internal references), not just assumed correct (using cross references).

Target-Object
The name of the object upon which the continuation must focus.

5.12 DS-DN

An instance of OM class **DS-DN** represents a name of a directory object.

An instance of this OM class has the OM attributes of its superclasses (*Object*, *Name*) and additionally the OM attributes listed in the table below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
RDNs	Object(DS-RDN)	-	0 or more	-

Table 5-8 OM Attributes of a DS-DN

RDNs
The sequence of RDNs that define the path through the DIT from its root to the object that the DS-DN denotes. The DS-DN of the root of the directory is the null name (no RDNs values). The order of the values is significant; the first value is closest to the root, and the last value is the RDN of the object.

5.13 DS-RDN

An instance of OM class **DS-RDN** is a relative distinguished name (RDN). An RDN uniquely identifies an immediate subordinate of an object whose entry appears in the DIT.

An instance of this OM class has the OM attributes of its superclasses (*Object*, *Relative-Name*) and additionally the OM attributes listed in the table below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
AVAs	Object(AVA)	-	1 or more	-

Table 5-9 OM Attributes of an RDN

AVAs
The AVAs that are marked by the DIB as components of the object's RDN. The assertions shall be true of the object but of none of its siblings, and the attribute types and values they contain shall appear in the object's directory entry. The order of the AVAs is not significant.

5.14 Entry-Information

An instance of OM class **Entry-Information** contains selected information from a single directory entry.

An instance of this OM class has the OM attributes of its superclass (*Object*, **Attribute-List**) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
From-Entry	Boolean	-	1	-
Object-Name	Object(Name)	-	1	-

Table 5-10 OM Attributes of an Entry-Info

The OM attribute **Attributes** is inherited from the superclass **Attribute-List**. It contains the information extracted from the directory entry of the target object. The type of each attribute requested and found will be present in the list, as will be its values if types and values were requested.

The OM class-specific OM attributes are:

From-Entry
Indicates whether the information was extracted from the specified object's entry, rather than from a copy of the entry.

Object-Name
The object's distinguished name.

5.15 Entry-Information-Selection

An instance of OM class **Entry-Information-Selection** identifies the information to be extracted from a directory entry.

An instance of this OM class has the OM attributes of its superclasses (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
All-Attributes	Boolean	-	1	true
Attributes-Selected	String(Object-Identifier)	-	0 or more	-
Info-Type	Enum(Info-Type)	-	1	types-and-values

Table 5-11 OM Attributes of an Entry-Info-Selection

All-Attributes
Indicates which attributes are of interest. Its value is one of:

- **false**
  meaning that information is requested about just those attributes listed in the OM attribute *Attributes-Selected*.
- **true**
  meaning that information is requested about all attributes in the directory entry. Any values of the OM attribute **Attributes-Selected** are ignored in this case.

Attributes-Selected
Lists the types of the attributes in the entry, from which information is to be extracted. The value of this OM attribute is only used if the value of **All-Attributes** is **false**. Supplying an empty list means that no attribute data will be returned, which can be used to verify the existence of an entry for a distinguished name.

Info-Type
Identifies what information is to be extracted from each identified attribute. Its value must be one of:

- TYPES_ONLY
  meaning that only the attribute types of the selected attributes in the entry are to be returned.
- TYPES_AND_VALUES
  meaning that both the attribute types and the attribute values of the selected attributes in the entry are to be returned.

5.16 Entry-Modification

An instance of OM class **Entry-Modification** describes a single modification to a specified attribute of a directory entry.

An instance of this OM class has the OM attributes of its superclasses (*Object*, **Attribute**) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Modification-Type	Enum(Modification-Type)	-	1	add-attribute

Table 5-12 OM Attributes of an Entry-Mod

The attribute type to be modified, and the associated values, are specified in the OM attributes **Attribute-Type** and **Attribute-Values**, which are inherited from the **Attribute** superclass.

Modification-Type
Identifies the type of the modification. Its value must be one of:

- ADD_ATTRIBUTE
  meaning that the specified attribute is absent and is to be added with the specified values.
- ADD_VALUES
  meaning that the specified attribute is present and the one or more specified values are to be added to it.
- REMOVE_ATTRIBUTE
  meaning that the specified attribute is present and is to be removed. Any values present in the OM attribute **Attribute-Values** are ignored.
- REMOVE_VALUES
  meaning that the specified attribute is present and the one or more specified values are to be removed from it.

5.17 Entry-Modification-List

An instance of OM class **Entry-Modification-List** comprises a sequence of changes to be made to a directory entry.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Changes	Object(Entry-Mod)	-	1 or more	-

Table 5-13 OM Attributes of an Entry-Modification-List

Changes
The modifications to be made, in the order specified, to the directory entry of the specified object.

5.18 Extension

An instance of OM class **Extension** denotes a standardised extension to the directory service set out in the standards. Such extensions will only be standardised in post-1988 versions of the standards.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Crit	Boolean	-	1	false
Identifier	Integer	-	1	-
Item-Parameters	any	-	1	-

Table 5-14 OM Attributes of an Extension

Critical
Its value is one of:

- **false**
  meaning that the originator will accept performance of the operation even if the extension is not available.
- **true**
  meaning that the originator must have the extended operation performed, or else have an error reported if it cannot be.

Identifier
Identifies the service extension. The values of this OM attribute will be assigned only by future versions of the standards.

Item-Parameters
This OM attribute supplies the parameters of the extension. Its syntax is determined by the particular Identifier.

5.19 Filter

An instance of OM class **Filter** is a basis for selecting or rejecting an object on the basis of information in its directory entry. At any point in time, an attribute filter has a value relative to every object. The value is false, true or undefined. The object is selected if, and only if, the filter's value is true.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Filter-Items	Object(Filter-Item)	-	0 or more	-
Filters	Object(Filter)	-	0 or more	-
Filter-Type	Enum(Filter-Type)	-	1	any

Table 5-15 OM Attributes of a Filter

A filter is a collection of simpler filters and elementary filter-items together with a Boolean operation. The filter value is undefined if and only if all the component Filters and Filter-Items are undefined. Otherwise, the filter has a Boolean value with respect to any directory entry, which can be determined by evaluating each of the nested components and combining their values using the Boolean operation. (Components whose value is undefined are ignored.)

Filter-Items
A collection of assertions, each relating to just one attribute of a directory entry.

Filters
A collection of simpler filters.

Filter-Type
The filter's type. Its value may be:

- **and**
  meaning that the filter is the logical conjunction of its components. The filter is **true** unless any of the nested filters or filter items is **false**. If there are no nested components, the filter is **true**.
- **or**
  meaning that the filter is the logical disjunction of its components. The filter is **false** unless any of the nested filters or filter items is **true**. If there are no nested components, the filter is **false**.
- **not**
  meaning that the result of this filter is reversed. There must be exactly one nested filter or filter item. The filter is **true** if the enclosed filter or filter item is **false**, and is **false** if the enclosed filter or filter item is **true**.

5.20 Filter-Item

An instance of OM class **Filter-Item** is a component of a **Filter**. It is an assertion about the existence or values of a single attribute type in a directory entry.

An instance of this OM class has the OM attributes of its superclasses (*Object*, **Attribute**) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Filter-Item-Type	Enum(Filter-Item-Type)	-	1	-
Final-Substring	String(*)	1 or more	0-1	-
Initial-Substring	String(*)	1 or more	0-1	-

Table 5-16 OM Attributes of a Filter-Item

Note that the OM attributes *Attribute-Type* and *Attribute-Values* are inherited from the superclass **Attribute**.

The value of the filter item is undefined if:

- the **Attribute-Type** is not known
- any of the **Attribute-Values** do not conform to the attribute syntax defined for that attribute type
- the **Filter-Item-Type** uses a matching rule that is not defined for the attribute syntax.

Access control restrictions may also cause the value to be undefined.

Filter-Item-Type
Identifies the type of the filter item and thereby the nature of the filter. Its value must be one of the following:

- **approximate-match**
  meaning that the filter is **true** if the directory entry contains at least one value of the specified type that is approximately equal to that specified (the meaning of *approximately equal* being implementation-dependent), and **false** otherwise.

  Rules for approximate matching are locally defined. For example, an approximate match might take into account spelling variations or employ phonetic comparison rules. In the absence of any such capabilities, a DSA should treat an approximate match as a test for equality.

  There must be exactly one value of the OM attribute *Attribute-Values*.

- **equality**
  meaning that the filter is **true** if the entry contains at least one value of the specified type that is equal to that specified (according to the equality matching rule in force), and **false** otherwise.

  There must be exactly one value of the OM attribute *Attribute-Values*.

- **greater-or-equal**
  meaning that the filter item is **true** if, and only if, at least one value of the attribute is greater than or equal to the supplied value (using the appropriate ordering algorithm).

  There must be exactly one value of the OM attribute *Attribute-Values*.

- **less-or-equal**
  meaning that the filter item is **true** if, and only if, at least one value of the attribute is less than or equal to the supplied value (using the appropriate ordering algorithm).

  There must be exactly one value of the OM attribute *Attribute-Values*.

- **present**
  meaning that the filter is **true** if the entry contains an attribute of the specified type, and **false** otherwise.

  Any values of the OM attribute *Attribute-Values* are ignored.

- **substrings**
  meaning that the filter is **true** if the entry contains at least one value of the specified attribute type that contains all of the specified substrings in the given order, and **false** otherwise.

  There can be any number of substrings given as values of the OM attribute *Attribute-Values*, including none. There can also be a substring in **Initial** and/or **Final**. The substrings shall be non-overlapping, but they may be separated from each other or from the ends of the attribute value by zero or more string elements. However, there must exist at least one attribute of type *Attribute-Values*, *Initial-Substring* or *Final-Substring*.

Final-Substring
If present, the substring that is to match the final portion of an attribute value in the entry. This attribute may only exist if the **Filter-Item-Type** is equal to **substrings**.

Initial-Substring
If present, the substring that is to match the initial portion of an attribute value in the entry. This attribute may only exist if the **Filter-Item-Type** is equal to **substrings**.

5.21 List-Info

An instance of OM class **List-Info** is a portion of the results of a *List*( ) function call.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Common-Results*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Object-Name	Object(Name)	-	0-1	-
Partial-Outcome-Qual	Object(Partial-Outcome-Qual)	-	0-1	-
Subordinates	Object(List-Info-Item)	-	0 or more	-

Table 5-17 OM Attributes of a List-Info

Object-Name
The distinguished name of the target object of the operation. It will be present if the OM attribute **Alias-Dereferenced**, inherited from the superclass *Common-Results*, is **true**.

Partial-Outcome-Qual
This OM attribute value is present if the list of subordinates is incomplete. The DSA or DSAs that provided this list did not complete the search for some reason. The partial outcome qualifier contains details of why the search was not completed, and which areas of the directory were not searched.

Subordinates
Information about zero or more subordinate objects identified by the *List*( ) function.

5.22 List-Info-Item

An instance of OM class **List-Info-Item** comprises details, returned by *List*( ), of a single subordinate object.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Alias-Entry	Boolean	-	1	-
From-Entry	Boolean	-	1	-
RDN	Object(Relative-Name)	-	1	-

Table 5-18 OM Attributes of a List-Info-Item

Alias-Entry
Indicates whether the object is an alias.

From-Entry
Indicates whether information about the object was obtained directly from its directory entry, rather than a copy of the entry.

RDN
The Relative Distinguished Name of the object. If this is the name of an alias entry (as indicated by **Alias-Entry**) it will not be dereferenced.

5.23 List-Result

An instance of OM class **List-Result** comprises the results of a successful call to the *List*( ) function.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
List-Info	Object(List-Info)	-	0-1	-
Uncorrelated-List-Info	Object(List-Result)	-	0 or more	-

Table 5-19 Attributes of a List-Result

No instance will contain values of both the OM attributes.

List-Info

The results of the *List*( ) function, or a portion thereof. If the directory entry whose subordinates are being listed has no subordinates, this attribute will have a single value which will have a 'subordinates' attribute with no values.

Uncorrelated-List-Info

When the DUA has requested a protection request of *signed*, the returned information may comprise a number of sets of results originating from and signed by different components of the directory. Implementations may reflect this structure by nesting **List-Result** OM objects as values of this OM attribute. Alternatively, they may collapse all results into a single value of the OM attribute *List-Info*.

5.24 Name

The OM class **Name** represents a name of an object in the directory, or a part of such a name.

It is an abstract class, which has the attributes of its superclass (*Object*) and no other OM attributes.

A name unambiguously distinguishes the object from all other objects whose entries appear in the DIT. However, an object may have more than one name, that is, a name need not be unique. A distinguished name is unique; there are no other distinguished names that identify the same object. A relative distinguished name is a part of a name, and only distinguishes the object from others that are its siblings. Most of the interface functions take a name argument, the value of which must be an instance of one of the subclasses of this OM class. Thus, this OM class serves to collect together all possible representations of names.

This specification defines one subclass of this OM class, and thus a single representation for names:

DS-DN
: which provides a representation for names, including distinguished names.

It is expected that vendors will define additional subclasses to provide alternative representations.

5.25 Operation-Progress

An instance of OM class **Operation-Progress** specifies the progress or processing state of a directory request.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Name-Resolution-Phase	Enum(Name-Resolution-Phase)	-	1	-
Next-RDN-To-Be-Resolved	Integer	-	0-1	-

Table 5-20 OM Attributes of an Operation-Progress

The target name mentioned below is the name upon which processing of the directory request is currently focussed.

Name-Resolution-Phase
Indicates what phase has been reached in handling the target name. Its value is one of:

- **completed**
  meaning that the DSA holding the target object has been reached.
- **not-started**
  meaning that a DSA has not so far been reached with a naming context containing the initial RDN(s) of the name.
- **proceeding**
  meaning that the initial part of the name has been recognised, though the DSA holding the target object has not yet been reached.

Next-RDN-To-Be-Resolved
Indicates to the DSA which of the RDNs in the target name is next to be resolved. It takes the form of an integer in the range one to the number of RDNs in the name. This OM attribute only has a value if the value of **Name-Resolution-Phase** is **proceeding**.

The constant **Operation-Not-Started** (DS_OPERATION_NOT_STARTED) may be used in the **Context** of an operation instead of an instance of this OM class.

5.26 Partial-Outcome-Qualifier

An instance of OM class **Partial-Outcome-Qualifier** explains to what extent the results of a call to the *List*( ) or *Search*( ) function are incomplete and why.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Limit-Problem	Enum(Limit-Problem)	-	0-1	-
Unavailable-Crit-Ext	Boolean	-	1	-
Unexplored	Object(Continuation-Reference)	-	0 or more	-

Table 5-21 OM Attributes of a Partial-Outcome-Qual

Limit-Problem
If present, explains, in whole or in part, why the results are partial. Its value is one of:

- **administrative-limit-exceeded**
  meaning that an administrative limit was reached,
- **size-limit-exceeded**
  meaning that the maximum number of objects specified as a service control was reached,
- **time-limit-exceeded**
  meaning that the maximum number of seconds specified as a service control was reached.

Unavailable-Critical-Extensions
If **true**, indicates that some part of the directory cannot provide a requested critical service extension. The user requested one or more standard service extensions, by including values of the OM attribute *Extensions* in the **Context** supplied for the operation, and further indicated some of them to be essential by setting the OM attribute *Critical* in the extension to be **true**. Some DSA or DSAs cannot perform some of the critical extensions. In general, it is not possible to determine which DSA could not perform which particular extension.

Unexplored
Identifies any regions of the directory that were left unexplored, in such a way that the directory request can be continued. Only continuation references within the scope specified by the **Scope-Of-Referral** service control are included.

5.27 Presentation-Address

An instance of OM class **Presentation-Address** is a presentation address of an OSI application entity, used for OSI communications with it.

An instance of this OM class has the OM attributes of its superclasses (*Object*, *Address*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
N-Addresses	String(Octet)	-	1 or more	-
P-Selector	String(Octet)	-	0-1	-
S-Selector	String(Octet)	-	0-1	-
T-Selector	String(Octet)	-	0-1	-

Table 5-22 OM Attributes of a Presentation-Address

N-Addresses
The network addresses of the application entity.

P-Selector
The presentation selector.

S-Selector
The session selector.

T-Selector
The transport selector.

5.28 Read-Result

An instance of OM class **Read-Result** comprises the result of a successful call to the *Read*( ) function.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Common-Results*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Entry	Object(Entry-Info)	-	1	-

Table 5-23 OM Attributes of a Read-Result

Entry
The information extracted from the directory entry of the target object.

5.29 Relative-Name

The OM class *Relative-Name* represents the Relative Distinguished Names of objects in the directory.

It is an abstract class, which has the attributes of its superclass (*Object*) and no other OM attributes.

A relative distinguished name (RDN) is a part of a name, and only distinguishes the object from others that are its siblings. This OM class serves to collect together all possible representations of RDNs. An argument of interface functions that is an RDN, or an OM attribute value that is an RDN, will be an instance of one of the subclasses of this OM class.

This specification defines one subclass of this OM class, and thus a single representation for RDNs:

DS-RDN
: which provides a representation for relative distinguished names.

It is expected that vendors will define additional subclasses to provide alternative representations.

5.30 Search-Information

An instance of OM class **Search-Info** is a portion of the result of a *Search*( ) function call.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Common-Results*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Entries	Object(Entry-Info)	-	0 or more	-
Object-Name	Object(Name)	-	0-1	-
Partial-Outcome-Qual	Object(Partial-Outcome-Qual)	-	0-1	-

Table 5-24 OM Attributes of a Search-Info

Entries
: Information about zero or more objects found by the *Search*( ) function that matched the given selection criteria.

Object-Name
: The distinguished name of the target object of the operation. It will be present if the OM attribute **Alias-Dereferenced**, inherited from the superclass *Common-Results*, is **true**.

Partial-Outcome-Qual
: This OM attribute value is only present if the list of entries is incomplete. The DSA or DSAs that provided this list did not complete the search for some reason. The partial outcome qualifier contains details of why the search was not completed, and which areas of the directory were not searched.

5.31 Search-Result

An instance of OM class **Search-Result** comprises the result of a successful call to the *Search*( ) function.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Search-Info	Object(Search-Info)	-	0-1	-
Uncorrelated-Search-Info	Object(Search-Result)	-	0 or more	-

Table 5-25 OM Attributes of a Search-Result

No instance will contain values of both the OM attributes.

Search-Info
The result of the *Search*( ) function, or a portion thereof.

Uncorrelated-Search-Info
When the DUA has requested a protection request of *signed*, the returned information may comprise a number of sets of results originating from and signed by different components of the directory. Implementations may reflect this structure by nesting **Search-Result** OM objects as values of this OM attribute. Alternatively, they may collapse all results into a single value of the OM attribute *Search-Info*.

5.32 Session

An instance of OM class **Session** identifies a particular link from the application program to a DUA.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
DSA-Address	Object(Address)	-	0-1	*local*[1]
DSA-Name	Object(Name)	-	0-1	*local*[1]
File-Descriptor	Integer	-	1	see below
Requestor	Object(Name)	-	0-1	-

[1]The default values of these OM attributes are locally-administered.

Table 5-26 OM Attributes of a Session

The **Session** collects together all the information that describes a particular directory interaction. The parameters that are to control such a session are set up in an instance of this OM class, which is then passed as an argument to *Bind*( ). This sets the OM attributes that describe the actual characteristics of this session, and starts the session. Such a started session must be passed as the first argument to each interface function. The result of modifying a started session is unspecified. Finally, *Unbind*( ) is used to terminate the session, after which the parameters can be modified and a new session started using the same instance, if required. Multiple concurrent sessions can be run, by using multiple instances of this OM class.

The OM attributes of a session are:

DSA-Address
Indicates the address of the default DSA named by **DSA-Name**.

DSA-Name
Indicates the distinguished name of the DSA that will be used by default to service directory requests.

File-Descriptor (optional functionality)
Indicates the file descriptor associated with the session. The file descriptor may be used in subsequent calls to vendor-specific system facilities to suspend the process (for example, System V *poll*( ) or BSD (Berkeley Source Distribution) *select*( )). Its use for any other purpose is unspecified.

If the implementation does not define any suitable suspension facilities, or if the session is not started, the value is **No-Valid-File-Descriptor** (DS_NO_VALID_FILE_DESCRIPTOR).

Requestor
The distinguished name of the user of this directory session.

Applications can assume that an object of OM class **Session**, created with default values of all its OM attributes, will work with all the interface functions. Local administrators should ensure that this is the case. Such a session can be created by passing the constant **Default-Session** (DS_DEFAULT_SESSION) as an argument to *Bind*( ).

Chapter 6

Errors

6.1 Introduction

This Chapter defines the errors that can arise in the use of the interface and describes the method used to report them.

Errors are reported to the application program by means of the Status that is the result of every function (it is the function result in the C language binding for most functions). A function that completes successfully returns the value **success** [DS_SUCCESS]. When a function is not successful, and has been given a session argument that refers to a valid workspace or (in the cases of *Bind*( ), *Shutdown*( ) and *Version*( )) has been given a valid workspace as a workspace argument, then it will return a private object in that workspace, and the private object will be of one of the following classes: *Error*, **Attribute-Error** or **Referral**. When one of the functions *Bind*( ), *Shutdown*( ) or *Version*( ) has been given an invalid workspace as a workspace argument, then it will return the error constant [DS_INVALID_WORKSPACE]. When a function is not successful under other circumstances, it will return the error constant [DS_NO_WORKSPACE].

The picture is more complicated for asynchronous operations, because they can fail at two stages; either before the remote operation is started, or during it. The first type is reported immediately in the status of the invoking function, whilst the second is returned as the **Operation-Status** result of a later call to *Receive-result*( ).

Errors are classified into ten OM classes. The standards (see references **ISO 9594**) classify errors into eight different kinds: **Abandoned**, **Abandon-Failed**, **Attribute-Error**, **Name-Error**, **Referral**, **Security-Error**, **Service-Error** and **Update-Error**. The **DirectoryBind** operation returns a **Security-Error** or a **Service-Error**. This interface never returns an **Abandoned** error. The interface also defines three more kinds of error: **Library-Error**, **Communications-Error** and **System-Error**. Each of these kinds of error is represented by an OM class, and these are detailed below in alphabetical order. All of them inherit the OM attribute *Problem* from their superclass *Error*, which is described first. The OM classes defined in this Chapter are part of the Directory Service package introduced in Section 5.1 on page 53.

In order to allow automatic connection management, the interface may not communicate with a DSA when *Bind*( ) is called, but may defer it until an enquiry or modification is requested. Because of this flexibility, all functions can return the same errors as *Bind*( ). For example, a read operation may return an authentication error because the connection was deferred until access was actually needed. Such errors may also arise in the course of following an automatic referral list, irrespective of the connection management policy.

6.2 OM Class Hierarchy

This Section depicts the hierarchical organisation of the OM classes defined in this Chapter and so indicates how OM attributes are inherited from superclasses. Subclassification is indicated by indentation, and the names of abstract OM classes are rendered in italics. Thus, for example, the concrete OM class **Attribute-Problem** is an immediate subclass of the abstract OM class *Error*, which in turn is an immediate subclass of the abstract OM class *Object*.

Object (defined in reference **XOM**)
— **Attribute-Error**
— **Continuation-Reference** (see Chapter 5)
  — **Referral**
— *Error*
  — **Abandon-Failed**
  — **Attribute-Problem**
  — **Communications-Error**
  — **Library-Error**
  — **Name-Error**
  — **Security-Error**
  — **Service-Error**
  — **System-Error**
  — **Update-Error**

The application program is not permitted to create or modify any instances of any of these OM classes. Also, this specification does not mandate that any OM classes are encodable using *OM-Encode*( ) and *OM-Decode*( ).

A **Referral** is not a real error, and is not a subclass of *Error*, though it is reported in the same way as a **Status** result. An **Attribute-Error**, also not a subclass of *Error*, is special because it may report several problems at once. Each one is reported in an **Attribute-Problem**, which is a subclass of *Error*.

6.3 Error

The OM class *Error* comprises the parameters common to all errors.

It is an abstract OM class, which has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Problem	Enum(Problem)	-	1	-

Table 6-1 OM Attributes of an Error

Details of errors are returned in an instance of a subclass of this OM class. Each such subclass represents a particular kind of error, and is one of: **Abandon-Failed**, **Attribute-Problem**, **Communications-Error**, **Library-Error**, **Name-Error**, **Security-Error**, **Service-Error**, **System-Error** or **Update-Error**.

The OM attributes of an *Error* are:

Problem

Gives details of the error. A number of possible values are defined, but implementations may define additional values. Implementations will not return other values for error conditions described in this Chapter. Each of the standard values listed below is described under the relevant error OM class:

administrative-limit-exceeded,
affects-multiple-DSAs,
alias-dereferencing-problem,
alias-problem,
attribute-or-value-already-exists,
bad-argument,
bad-class,
bad-context,
bad-name,
bad-session,
bad-workspace,
busy,
cannot-abandon,
chaining-required,
communications-problem,
constraint-violation,
dit-error,
entry-already-exists,
inappropriate-authentication,
inappropriate-matching,
insufficient-access-rights,
invalid-attribute-syntax,
invalid-attribute-value,
invalid-credentials,
invalid-reference,
invalid-signature,
loop-detected,
miscellaneous,

missing-type,
mixed-synchronous,
naming-violation,
no-information,
no-such-attribute-or-value,
no-such-object,
no-such-operation,
not-allowed-on-RDN,
not-allowed-on-non-leaf,
not-supported,
object-class-modification-prohibited,
object-class-violation,
out-of-scope,
protection-required,
system error
time-limit-exceeded,
too-late,
too-many-operations,
too-many-sessions,
unable-to-proceed,
unavailable,
unavailable-critical-extension,
undefined-attribute-type,
unwilling-to-perform.

6.4 Abandon_failed

An instance of OM class **Abandon-Failed** reports a problem encountered during an attempt to abandon an operation.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and no additional OM attributes.

The following OM attribute is inherited from the superclass *Error*.

Problem
Identifies the problem. Its value is one of:

- **cannot-abandon**
  meaning that an attempt was made to abandon an operation for which this is prohibited, or the abandon could not be performed.
- **no-such-operation**
  meaning that the directory has no knowledge of the operation that is to be abandoned.
- **too-late**
  meaning that the operation is already completed, successfully or erroneously.

6.5 Attribute-Error

An instance of OM class **Attribute-Error** reports an attribute-related directory error.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Object-Name	Object(Name)	-	1	-
Problems	Object(Attribute-Problem)	-	1 or more	-

Table 6-2 OM Attributes of an Attribute-Error

Object-Name
The name of the directory entry to which the operation was being applied when the failure occurred.

Problems
Documents the attribute-related problems encountered. Uniquely, an **Attribute-Error** can report several problems at once. All are related to the above object.

6.6 Attribute-Problem

An instance of OM class **Attribute-Problem** documents one attribute-related problem encountered while performing an operation as requested on a particular occasion.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attribute-Type	String(Object-Identifier)	-	1	-
Attribute-Value	any	-	0-1	-

Table 6-3 OM Attributes of an Attribute-Problem

Attribute-Type
Identifies the type of the attribute with which the problem is associated.

Attribute-Value
The attribute value with which the problem is associated. Its syntax is determined by the value of **Attribute-Type**. This OM attribute shall be present if necessary to avoid ambiguity.

The following OM attribute is inherited from the superclass *Error*.

Problem
Identifies the problem. Its value is one of:

- **attribute-or-value-already-exists**
  meaning that an attempt was made to add an attribute or value that is already present in the directory entry in question.
- **constraint-violation**
  meaning that the attribute or attribute value does not conform to the constraints imposed by the standards (see reference **ISO 9594**) or by the attribute definition (for example, the value exceeds its upper bound).
- **inappropriate-matching**
  meaning that an attempt was made to use a matching rule that is not defined for the attribute type.
- **invalid-attribute-syntax**
  meaning that a value presented as an argument does not conform to the attribute syntax of the attribute type.
- **no-such-attribute-or-value**
  meaning that the specified attribute or value was not found in the directory entry in question.

  This is only reported by a *Read*( ) or *Search*( ) operation if an explicit list of attributes is specified by the *Selection* argument, but none of them are present in the entry.
- **undefined-attribute-type**
  meaning that the attribute type, which was supplied as an argument to *Add-Entry*( ) or *Modify-Entry*( ), is undefined.

6.7 Communications-Error

An instance of OM class **Communications-Error** reports an error occurring in the other OSI services supporting the Directory Service.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and no additional OM attributes.

Communications errors include those arising in Remote Operation, Association Control, Presentation, Session and Transport.

The following OM attribute is inherited from the superclass *Error*:

Problem
Its value is **communications-problem**.

6.8 Library-Error

An instance of OM class **Library-Error** reports an error detected by the interface function library.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and no additional OM attributes.

Each function has several possible errors which can be detected by the library itself, and which are returned directly by the subroutine. These errors occur when the library itself is incapable of performing an action, submitting a service request, or deciphering a response from the directory.

The following OM attribute is inherited from the superclass *Error*:

Problem

Identifies the particular library error that has occurred. The **ERRORS** section of each function description lists just those which that function can return. Its value is one of:

- **bad-argument**
  meaning that a bad argument (other than name) was supplied. Use of an instance of OM class **Attribute** with no values of the OM attribute *Attribute-Values* as an input argument to a directory service function will result in this error (because directory attributes always have at least one value).
- **bad-class**
  meaning that the OM class of an argument is not supported for this operation.
- **bad-context**
  meaning that an invalid context argument was supplied.
- **bad-name**
  meaning that an invalid name argument was supplied.
- **bad-session**
  meaning that an invalid session was supplied.
- **bad-workspace**
  meaning that a function was passed a session argument that was associated with one workspace and was also passed another argument that had been created in a different workspace.
- **miscellaneous**
  meaning that a miscellaneous error occurred in interacting with the directory. This error will be returned if the interface cannot clear a transient system error by retrying the affected system call.
- **missing-type**
  meaning that the attribute type was not included in an attribute-value-assertion passed as part of a distinguished name argument.
- **mixed-synchronous**
  meaning that an attempt to start a synchronous operation was made whilst there were outstanding asynchronous operations.
- **not-supported**
  meaning that an attempt was made to use optional functionality, which is not available in this implementation.

- **too-many-operations**
  meaning that no more directory operations can be performed until at least one asynchronous operation has completed.
- **too-many-sessions**
  meaning that no more directory sessions can be started.

6.9 Name-Error

An instance of OM class **Name-Error** reports a name-related directory error.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Matched	Object(Name)	-	1	-

Table 6-4 OM Attributes of a Name-Error

Matched
The initial portion, up to but excluding the first RDN that is unrecognised, of the name that was supplied or of the name resulting from dereferencing an alias. It names the lowest entry (object or alias) in the DIT that was matched.

The following OM attribute is inherited from the superclass *Error*.

Problem
Identifies the cause of the failure. Its value is one of:

- **alias-dereferencing-problem**
  meaning that an alias was encountered where an alias is not allowed. An alias was encountered in a modification operation or when the **Dont-Dereference-Alias** service control was set, or one alias points to another alias.
- **alias-problem**
  meaning that an alias has been dereferenced which names an object that does not exist (that is, for which no directory entry can be found).
- **invalid-attribute-value**
  meaning that the attribute value in an AVA in an RDN in the name does not conform to the attribute syntax prescribed for the attribute type in the AVA. (This problem is called *invalidAttributeSyntax* in the standards, but that name is used only for an **Attribute-Problem** in this interface).
- **no-such-object**
  meaning that the specified name does not match the name of any object in the directory.

6.10 Referral

An instance of OM class **Referral** reports failure to perform an operation and redirects the requestor to one or more access points better equipped to perform it.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, **Continuation-Reference**) and no additional OM attributes.

The referral is a continuation reference by means of which the operation may be progressed.

6.11 Security-Error

An instance of OM class **Security-Error** reports a security-related directory error.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and no additional OM attributes.

The following OM attribute is inherited from the superclass *Error*:

Problem
Identifies the cause of the failure. Its value is one of:

- **inappropriate-authentication**
  meaning that the level of security attached to the requestor's credentials is inconsistent with the level of protection requested (for example, simple credentials were supplied while strong credentials were required).
- **insufficient-access-rights**
  meaning that the requestor does not have permission to perform the operation. A *Read*( ) operation will only return this error when access rights preclude the reading of all requested attribute values.
- **invalid-credentials**
  meaning that the requestor's credentials are invalid.
- **invalid-signature**
  meaning that the signature affixed to the request is invalid.
- **no-information**
  meaning that the request produced a security error for which no other information is available.
- **protection-required**
  meaning that the directory is unwilling to perform the operation, because it was unsigned.

6.12 Service-Error

An instance of OM class **Service-Error** reports a directory error related to the provision of service.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and no additional OM attributes.

The following OM attribute is inherited from the superclass *Error*:

Problem
Identifies the cause of the failure. Its value is one of:

- **administrative-limit-exceeded**
  meaning that the operation could not be performed within the administrative constraints on the directory and no partial results are available.
- **busy**
  meaning that some part of the directory is temporarily too busy to perform the operation (but may be able to do so after a short while).
- **chaining-required**
  meaning that chaining is required to perform the operation but is prohibited by the **Chaining-Prohibited** service control.
- **dit-error**
  meaning that an inconsistency has been detected in the DIT, which may be localised to a particular entry or set of entries.
- **invalid-reference**
  meaning that the DSA was unable to perform the request as directed (via **Operation-Progress** in the **Context**). This may be because of an invalid referral.
- **loop-detected**
  meaning that a DSA detected a loop within the directory.
- **out-of-scope**
  meaning that the directory cannot provide a referral or partial outcome qualifier within the required scope.
- **time-limit-exceeded**
  meaning that the operation could not be performed within the time specified by the **time-limit** service control, and no partial results are available.
- **unable-to-proceed**
  meaning that a DSA without administrative authority over a particular naming context was asked to resolve a name in that context.
- **unavailable**
  meaning that some part of the directory is not currently available.
- **unavailable-critical-extension**
  meaning that one or more critical extensions were requested but not available.
- **unwilling-to-perform**
  meaning that some part of the directory is not willing to perform the operation because it requires excessive resources, or because doing so would violate administrative policy.

6.13 System-Error

An instance of OM class **System-Error** reports an error occurring in the underlying operating system.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and no additional OM attributes, though there may be additional implementation-defined OM attributes.

The following OM attribute is inherited from the superclass *Error*:

Problem

Identifies the cause of the failure. Its value is the same as that of *errno* defined in the C language.

The standard names of system errors are defined in the **<errno.h>** header, which is defined in the **XPG4 System Interfaces and Headers** document (see reference **XPG4**). and additional names may be implementation-defined.

If a transient error occurs ([EINTR] or [EAGAIN]), implementations will retry the affected operation and do not report these errors. If such an error persists they may report the **Library-Error** ''miscellaneous'' or an implementation-defined library error.

6.14 Update-Error

An instance of OM class **Update-Error** reports a directory error peculiar to a modification operation.

An application is not permitted to create or modify instances of this OM class. An instance of this OM class has the OM attributes of its superclasses (*Object*, *Error*) and no additional OM attributes.

The following OM attribute is inherited from the superclass *Error*:

Problem
Identifies the cause of the failure. Its value is one of:

- **affects-multiple-DSAs**
  meaning that the modification would affect several DSAs, which is not permitted. Local agreement between DSAs may allow modifications which do affect multiple DSAs, such as adding entries whose immediate superior entry is in a different DSA. This problem will not be reported in such cases.
- **entry-already-exists**
  meaning that the name passed to *Add-Entry*( ) already exists.
- **naming-violation**
  meaning that the modification would leave the DIT improperly structured. That is, it would add an entry as the subordinate of an alias, or in a region of the DIT not permitted to a member of its object class, or would define an RDN that includes a forbidden attribute type.
- **not-allowed-on-non-leaf**
  meaning that the modification would be to an interior node of the DIT (and such a modification is prohibited).
- **not-allowed-on-RDN**
  meaning that the modification would alter an object's RDN.
- **object-class-modification-prohibited**
  meaning that the modification would alter an entry's Object Class attribute.
- **object-class-violation**
  meaning that the modification would leave a directory entry inconsistent with its Object Class definition.

Chapter 7

Directory Class Definitions

7.1 Introduction

The standards define a number of attribute types (known as the *selected attribute types*), attribute syntaxes, attribute sets and object classes (known as the *selected object classes*). These definitions allow the creation and maintenance of directory entries for a number of common objects, so that the representation of all such objects will be the same throughout the directory. The definitions are chiefly in the ISO Standards 9594-6 and 9594-7 with additional material in 9594-1 and 9594-8 (see **Referenced Documents**). They include such objects as **Country**, **Person** and **Organisation**.

This Chapter sets out names for each of these items, and defines OM classes to represent those which are not represented directly by OM syntaxes. The values of attributes in the directory are not restricted to those discussed in this Chapter, and new attribute types and syntaxes may be created at any time. Implementations are likely to add additional definitions. Section 3.5.1 on page 20 defines how the values of other syntaxes are represented in the interface.

The constants and OM classes defined in this Chapter are additional to those presented in Chapter 5, since they are not essential to the working of the interface, but instead allow directory entries to be utilised. The definitions are further divided into three packages, each of which is optionally supported.

One of the packages is called the **Basic Directory Contents Package (BDCP)** and contains all of the definitions except those concerned with strong authentication. These latter definitions are collected in the **Strong Authentication Package (SAP)**. A third optional package is the **MHS Directory User Package (MDUP)**, which contains definitions to support the use of the Directory by 1988 X.400 User Agents and MTAs for the purposes of name resolution, DL expansion and capability assessment. The definitions are based upon the attribute types and syntaxes specified in X.402, Annex A.

The object identifier associated with the BDCP is
{iso(1) member-body(2) us(840) IEEE-P1224.2(10014) bdcp(1)}
(with the encoding *"\x2a\x86\x48\xce\x1e\x1"*). This identifier is represented by the constant
Basic-Directory-Contents-Package (DS_BASIC_DIRECTORY_CONTENTS_PACKAGE)
The C constants associated with this package are in the **<xdsbdcp.h>** header.

The object identifier associated with the SAP is
{iso(1) member-body(2) us(840) IEEE-P1224.2(10014) sap(2)}
(with the encoding *"\x2a\x86\x48\xce\x1e\x2"*). This identifier is represented by the constant
Strong-Authentication-Package (DS_STRONG_AUTHENTICATION_PACKAGE)
The C constants associated with this package are in the **<xdssap.h>** header.

The object identifier associated with the MDUP is
{iso(1) member-body(2) us(840) IEEE-P1224.2(10014) mdup(3)}
(with the encoding *"\x2a\x86\x48\xce\x1e\x3"*). This identifier is represented by the constant
MHS-Directory-User-Package (DS_MHS_DIRECTORY_USER_PACKAGE)
The C constants associated with this package are in the **<xdsmdup.h>** header.

The concepts and notation used are introduced in Section 1.3 on page 3, and Section 5.1 on page 53, and fully explained in the referenced **XOM** Specification. A complete explanation of the meaning of the attributes and object classes is not given, since this is outside the scope of this specification. The purpose here is simply to present the representation of these items in the interface.

The selected attribute types are presented first, followed by the selected object classes, and finally the OM class hierarchy and OM class definitions required to support the selected attribute types.

7.2 Selected Attribute Types

This Section presents the attribute types defined in the standards for use in directory entries. Each directory entry is made up of a number of attributes, each of which comprises an attribute type together with one or more attribute values. The form of each value of an attribute is determined by the attribute syntax associated with the attribute's type.

In the interface, attributes appear as instances of OM class **Attribute**, with the attribute type represented as the value of the OM attribute *Attribute-Type* and the attribute value(s) represented as the value(s) of the OM attribute *Attribute-Values*. Each attribute type has an object-identifier, assigned in the standards, which is the value of the OM attribute *Attribute-Type*. These object-identifiers are represented in the interface by constants with the same name as the (directory) attribute, prefixed with **A-** for ease of identification. (Consequently, the C constants start with DS_A_.)

Several of the attribute types are defined in the 1992 version of the Standards to have ASN.1 syntax `DirectoryString`. This is a choice of `TeletexString`, `PrintableString` and `UniversalString`. In these cases, the values of the corresponding *Attribute-Values* OM attributes can have syntaxes `String(Teletex)`, `String(Printable)` or `String(Universal)`. This is indicated by describing their syntaxes as `String(Directory)`.

This Section contains two tables. The first tabulates the names of the attribute types defined in the standards, together with the BER encoding of the object identifiers associated with each of them. The second tabulates the names of the attribute types, together with the OM Value Syntax used in the interface to represent values of that attribute type. The table also includes the range of lengths permitted for the string types, an indication of whether the attribute can be multi-valued, and an indication of which matching rules are provided for the syntax. Following the table is a brief description of each attribute.

The standards define matching rules that are used for deciding whether two values are equal (E), for ordering two values (O), and for identifying one value as a substring of another (S) in directory operations. Specific matching rules are given below for certain attributes. Additionally, the following three general rules apply as indicated.

- For all attribute values whose syntax is `String(Numeric)`, `String(Printable)` or `String(Teletex)` , differences caused by the presence of spaces preceding the first printing character, spaces following the last printing character, and more than one consecutive space anywhere within the values, shall be considered insignificant.
- For all attribute values whose syntax is `String(Teletex)`, differences in the case of alphabetical characters shall be considered insignificant.
- For all attribute values whose syntax is indicated as `String(Directory)`, differences in the case of alphabetical characters shall be considered insignificant and, if the strings being compared are of different syntax, the comparison shall proceed as normal so long as the corresponding characters are in both character sets, but shall fail otherwise.

Table 7-1 Object Identifiers for Selected Attribute Types

Note: The third and fourth columns of this table contain the decimal and hexadecimal values respectively of the octets of the BER encoding of the object identifier. All BDCP and SAP package object identifiers stem from the root {joint-iso-ccitt ds(5) attributeType(4)}. MDUP object identifiers stem from the root {joint-iso-ccitt mhs-motis(6) arch(5) at(2)}.

Package	Attribute Type	Object Identifier BER	
		decimal	hex
BDCP	**A-Aliased-Object-Name**	85, 4, 1	\x55\x04\x01
SAP	**A-Authority-Revoc-List**	85, 4, 38	\x55\x04\x26
BDCP	**A-Business-Category**	85, 4, 15	\x55\x04\x0F
SAP	**A-CA-Cert**	85, 4, 37	\x55\x04\x25
SAP	**A-Cert-Revoc-List**	85, 4, 39	\x55\x04\x27
BDCP	**A-Common-Name**	85, 4, 3	\x55\x04\x03
BDCP	**A-Country-Name**	85, 4, 6	\x55\x04\x06
SAP	**A-Cross-Cert-Pair**	85, 4, 40	\x55\x04\x28
MDUP	**A-Deliverable-Content-Length**	86, 5, 2, 0	\x56\x05\x02\x00
MDUP	**A-Deliverable-Content-Types**	86, 5, 2, 1	\x56\x05\x02\x01
MDUP	**A-Deliverable-EITs**	86, 5, 2, 2	\x56\x05\x02\x02
MDUP	**A-DL-Members**	86, 5, 2, 3	\x56\x05\x02\x03
MDUP	**A-DL-Submit-Permissions**	86, 5, 2, 4	\x56\x05\x02\x04
BDCP	**A-Description**	85, 4, 13	\x55\x04\x0D
BDCP	**A-Dest-Indicator**	85, 4, 27	\x55\x04\x1B
BDCP	**A-Facsimile-Telephone-Number**	85, 4, 23	\x55\x04\x17
BDCP	**A-International-ISDN-Number**	85, 4, 25	\x55\x04\x19
BDCP	**A-Knowledge-Information**	85, 4, 2	\x55\x04\x02
BDCP	**A-Locality-Name**	85, 4, 7	\x55\x04\x07
BDCP	**A-Member**	85, 4, 31	\x55\x04\x1F
MDUP	**A-Message-Store**	86, 5, 2, 5	\x56\x05\x02\x05
BDCP	**A-Object-Class**	85, 4, 0	\x55\x04\x00
MDUP	**A-OR-Address**	86, 5, 2, 6	\x56\x05\x02\x06
BDCP	**A-Organization-Name**	85, 4, 10	\x55\x04\x0A
BDCP	**A-Organizational-Unit-Name**	85, 4, 11	\x55\x04\x0B
BDCP	**A-Owner**	85, 4, 32	\x55\x04\x20
BDCP	**A-Phys-Deliv-Off-Name**	85, 4, 19	\x55\x04\x13
BDCP	**A-Post-Office-Box**	85, 4, 18	\x55\x04\x12
BDCP	**A-Postal-Address**	85, 4, 16	\x55\x04\x10
BDCP	**A-Postal-Code**	85, 4, 17	\x55\x04\x11
BDCP	**A-Pref-Deliv-Method**	85, 4, 28	\x55\x04\x1C
MDUP	**A-Pref-Deliv-Methods**	86, 5, 2, 7	\x56\x05\x02\x07
BDCP	**A-Presentation-Address**	85, 4, 29	\x55\x04\x1D
BDCP	**A-Registered-Address**	85, 4, 26	\x55\x04\x1A
BDCP	**A-Role-Occupant**	85, 4, 33	\x55\x04\x21
BDCP	**A-Search-Guide**	85, 4, 14	\x55\x04\x0E
BDCP	**A-See-Also**	85, 4, 34	\x55\x04\x22
BDCP	**A-Serial-Number**	85, 4, 5	\x55\x04\x05
BDCP	**A-State-Or-Province-Name**	85, 4, 8	\x55\x04\x08
BDCP	**A-Street-Address**	85, 4, 9	\x55\x04\x09
BDCP	**A-Support-Applic-Context**	85, 4, 30	\x55\x04\x1E
MDUP	**A-Supp-Auto-Actions**	86, 5, 2, 8	\x56\x05\x02\x08

MDUP	**A-Supp-Content-Types**	86, 5, 2, 9	\x56\x05\x02\x09
MDUP	**A-Supp-Opt-Attributes**	86, 5, 2, 10	\x56\x05\x02\x0a
BDCP	**A-Surname**	85, 4, 4	\x55\x04\x04
BDCP	**A-Telephone-Number**	85, 4, 20	\x55\x04\x14
BDCP	**A-Teletex-Term-Ident**	85, 4, 22	\x55\x04\x16
BDCP	**A-Telex-Number**	85, 4, 21	\x55\x04\x15
BDCP	**A-Title**	85, 4, 12	\x55\x04\x0C
SAP	**A-User-Cert**	85, 4, 36	\x55\x04\x24
BDCP	**A-User-Password**	85, 4, 35	\x55\x04\x23
BDCP	**A-X121-Address**	85, 4, 24	\x55\x04\x18

Table 7-2 Representation of Values for Selected Attribute Types

[1] As permitted by ISO 3166.
[2] As permitted by Recommendations F.1 and F.31.
[3] As permitted by E.164.
[4] As permitted by E.123 (for example, +44 582 10101).
[5] As permitted by X.121.

Attribute Type	OM Value Syntax	Value Length	Multi-Valued	Matching Rules
A-Aliased-Object-Name	`Object(Name)`	-	no	E
A-Authority-Revoc-List	`Object(Revocation-List)`	-	yes	
A-Business-Category	`String(Directory)`	1-128	yes	E, S
A-CA-Cert	`Object(Certificate)`	-	yes	
A-Cert-Revoc-List	`Object(Revocation-List)`	-	yes	
A-Common-Name	`String(Directory)`	1-64	yes	E, S
A-Country-Name	`String(Printable)`[1]	2	no	E
A-Cross-Cert-Pair	`Object(Certificate-Pair)`	-	yes	
A-Deliverable-Content-Length	`Integer`	-	no	
A-Deliverable-Content-Types	`String(Object-Identifier)`	-	yes	
A-Deliverable-EITs	`String(Object-Identifier)`	-	yes	
A-Description	`String(Directory)`	1-1024	yes	E, S
A-Dest-Indicator	`String(Printable)`[2]	1-128	yes	E, S
A-DL-Members	`Object(OR-Name)`	-	yes	
A-DL-Submit-Permissions	`Object(DL-Submit-Permission)`	-	yes	
A-Facsimile-Telephone-Number	`Object(Facsimile-Telephone-Number)`	-	yes	
A-International-ISDN-Number	`String(Numeric)`[3]	1-16	yes	
A-Knowledge-Information	`String(Directory)`	-	yes	E, S
A-Locality-Name	`String(Directory)`	1-128	yes	E, S
A-Member	`Object(Name)`	-	yes	E
A-Message-Store	`Object(DS-DN)`	-	no	
A-Object-Class	`String(Object-Identifier)`	-	yes	E
A-OR-Addresses	`Object(OR-Address)`	-	yes	
A-Organization-Name	`String(Directory)`	1-64	yes	E, S
A-Organizational-Unit-Name	`String(Directory)`	1-64	yes	E, S
A-Owner	`Object(Name)`	-	yes	E
A-Phys-Deliv-Off-Name	`String(Directory)`	1-128	yes	E, S
A-Post-Office-Box	`String(Directory)`	1-40	yes	E, S
A-Postal-Address	`String(Directory)`	-	yes	E

A-Postal-Code	`String(Directory)`	1-40	yes	E, S
A-Pref-Deliv-Method	`Enum(Pref-Deliv-Method)`	-	yes	
A-Pref-Deliv-Methods	`Enum(Delivery-Mode)`	-	no	E
A-Presentation-Address	`Object(Presentation-Address)`	-	no	E
A-Registered-Address	`Object(Postal-Address)`	-	yes	
A-Role-Occupant	`Object(Name)`	-	yes	E
A-Search-Guide	`Object(Search-Guide)`	-	yes	
A-See-Also	`Object(Name)`	-	yes	E
A-Serial-Number	`String(Printable)`	1-64	yes	E, S
A-State-Or-Province-Name	`String(Directory)`	1-128	yes	E, S
A-Street-Address	`String(Directory)`	1-128	yes	E, S
A-Support-Applic-Context	`String(Object-Identifier)`	-	yes	E
A-Supp-Auto-Actions	`String(Object-Identifier)`	-	yes	
A-Supp-Content-Types	`String(Object-Identifier)`	-	yes	
A-Supp-Opt-Attributes	`String(Object-Identifier)`	-	yes	
A-Surname	`String(Directory)`	1-64	yes	E, S
A-Telephone-Number	`String(Printable)`[4]	1-32	yes	E, S
A-Teletex-Term-Ident	`Object(Teletex-Term-Ident)`	-	yes	
A-Telex-Number	`Object(Telex-Number)`	-	yes	
A-Title	`String(Directory)`	1-64	yes	E, S
A-User-Cert	`Object(Certificate)`	-	yes	
A-User-Password	`String(Octet)`	0-128	yes	
A-X121-Address	`String(Numeric)`[5]	1-15	yes	E, S

Throughout the descriptions that follow, the term *object* denotes the (directory) object whose (directory) entry contains the corresponding (directory) attributes.

A-Aliased-Object-Name
This attribute occurs only in alias entries. It gives the distinguished name of the object that is provided with an alias by the entry in which this attribute occurs. An alias is an alternative to an object's distinguished name. Any object may (but need not) have one or more aliases. The directory is said to dereference an alias whenever it replaces the alias, during name processing, with the distinguished name associated with it by means of this attribute.

A-Authority-Revocation-List
This attribute occurs only in entries that describe a Certification Authority (CA). It lists all the certificates issued to any of the CAs known to this CA, and later revoked. The values of this OM attribute shall be signed by the CA.

A-Business-Category
Descriptions of the businesses in which the object is engaged.

A-CA-Certificate
The certificates assigned to the object, which shall be a Certification Authority (CA).

A-Certificate-Revocation-List
This attribute occurs only in entries that describe a Certification Authority (CA). The certificates issued by this CA and later revoked. The values of this OM attribute shall be signed by the CA.

A-Common-Name
Names by which the object is commonly known in the context defined by its position in the DIT. They may conform to the naming convention of the country or culture with which the object is associated. They may be ambiguous.

A-Country-Name
Identifies the country in which the object is located or with which it is associated in some other important way. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

A-Cross-Certificate-Pair
One or two certificates, held in the entry of a Certification Authority (CA). The first certificate is that of one CA, guaranteed by a second CA, whilst the second certificate is that of the second CA, guaranteed by the first CA.

A-Deliverable-Content-Length
This attribute identifies the maximum content length of the messages whose delivery a user will accept.

A-Deliverable-Content-Types
This attribute identifies the content types of the messages whose delivery a user will accept.

A-Deliverable-EITs
This attribute identifies the EITs of the messages whose delivery a user will accept.

A-Description
Informational descriptions of the object.

A-Destination-Indicator
Country-city pairs by means of which the object can be reached via the public telegram service. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

A-DL-Members
This attribute identifies a DL's members.

A-DL-Submit-Permissions
This attribute identifies the users and DLs that may submit messages to a DL.

A-Facsimile-Telephone-Number
Telephone numbers for facsimile terminals (and optionally their parameters) by means of which the object can be reached or with which it is associated in some other important way.

A-International-ISDN-Number
International ISDN numbers by means of which the object can be reached or with which it is associated in some other important way. The matching rules require that differences caused by the presence of spaces are considered insignificant.

A-Knowledge-Information
This attribute occurs only in entries that describe a DSA. It provides a human-intelligible accumulated description of the directory knowledge possessed by the DSA.

A-Locality-Name
Identifies geographical areas or localities. When used as part of a directory name, it specifies the localities in which the object is located or with which it is associated in some other important way.

A-Member
Names of objects that are considered members of the present object (which might be a distribution list for electronic mail, for example).

A-Message-Store
This attribute identifies a user's MS by name.

A-Object-Class
Identifies the object classes to which the object belongs, and also identifies their superclasses. All such object classes that have object identifiers assigned to them are present, except that object class Top need not (but may) be present as long as some other value is present. This attribute must be present in every entry and may not be modified. It is discussed further in the next section.

A-OR-Addresses
This attribute specifies a user's or DL's O/R addresses.

A-Organization-Name
Identifies organisations. When used as part of a directory name, it specifies an organisation with which the object is affiliated. Several values may identify the same organisation in different ways.

A-Organizational-Unit-Name
Identifies organisational units. When used as part of a directory name, it specifies an organisational unit with which the object is affiliated. The units are understood to be parts of the organisation that the Organization-Name attribute denotes. Several values may identify the same unit in different ways.

A-Owner
The names of objects that have responsibility for the object.

A-Physical-Delivery-Office-Name
The names of cities, towns or villages, etc., that contain physical delivery offices through which the object can take delivery of physical mail.

A-Post-Office-Box
Identifies post office boxes at which the object can take delivery of physical mail. This information also appears as part of the Postal-Address attribute, if it is present.

A-Postal-Address
Postal addresses at which the object can take delivery of physical mail. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

A-Postal-Code
Postal codes assigned to areas or buildings through which the object can take delivery of physical mail. This information also appears as part of the Postal-Address attribute, if it is present.

A-Preferred-Delivery-Method
The object's preferred methods for communicating with it, ordered with the most preferred first. Each value is one of:

- **any-delivery-method**, meaning the object has no preference.
- **g3-facsimile-delivery**, meaning via Group 3 facsimile.
- **g4-facsimile-delivery**, meaning via Group 4 facsimile.
- **ia5-terminal-delivery**, meaning via IA5 text.
- **mhs-delivery**, meaning via X.400.
- **physical-delivery**, meaning via the postal or other physical delivery system.
- **telephone-delivery**, meaning via telephone.
- **teletex-delivery**, meaning via teletex.

- **telex-delivery**, meaning via telex.
- **videotex-delivery**, meaning via videotex.

A-Preferred-Delivery-Methods
This attribute identifies, in order of decreasing preference, the methods of delivery a user prefers.

A-Presentation-Address
The (OSI) presentation address of the object, which shall be an (OSI) application entity. The matching rule for a presented value to match a value stored in the directory is that the P-Selector, S-Selector and T-Selector of the two Presentation-Addresses must be equal, and the N-Addresses of the presented value must be a subset of those of the stored value.

A-Registered-Address
Mnemonics by means of which the object can be reached via the public telegram service (according to F.1). A mnemonic identifies an object in the context of a particular city, and is registered in the country containing the city. The matching rules require that differences in the case of alphabetical characters are considered insignificant.

A-Role-Occupant
This attribute occurs only in entries that describe an organisational role. It gives the names of objects that fulfill the organisational role.

A-Search-Guide
Criteria that can be used to build filters for conducting searches in which the object is the base object.

A-See-Also
Names of objects that represent other aspects of the real-world object that the present object represents.

A-Serial-Number
Serial numbers of a device.

A-State-Or-Province-Name
Specifies a state or province. When used as part of a directory name, it identifies states, provinces or other geographical regions in which the object is located or with which it is associated in some other important way.

A-Street-Address
A street address identifies a site for the local distribution and physical delivery of mail. When used as part of a directory name, it identifies the Street address (for example, street name and house number) at which the object is located or with which it is associated in some other important way.

A-Supported-Application-Context
This attribute occurs only in entries that describe an (OSI) application entity. It identifies (OSI) application contexts supported by the object.

A-Supported-Automatic-Actions
This attribute identifies the automatic actions that an MS fully supports.

A-Supported-Content-Types
This attribute identifies the content types of the messages whose syntax and semantics a MS fully supports.

A-Supported-Optional-Attributes
This attribute identifies the optional attributes that an MS fully supports.

A-Surname
This attribute occurs only in entries that describe individuals. The surname by which the individual is commonly known, normally inherited from the individual's parent(s) or taken-on marriage, as determined by the custom of the country or culture with which the individual is associated.

A-Telephone-Number
Identifies telephones by means of which the object can be reached or with which it is associated in some other important way. The matching rules require that differences caused by the presence of spaces and dashes are considered insignificant.

A-Teletex-Terminal-Identifier
Descriptions of teletex terminals by means of which the object can be reached or with which it is associated in some other important way.

A-Telex-Number
Descriptions of telex terminals by means of which the object can be reached or with which it is associated in some other important way.

A-Title
Identifies positions or functions of the object within its organisation.

A-User-Certificate
The user certificates assigned to the object, which may be any user certificate including a CA certificate.

A-User-Password
The passwords assigned to the object.

A-X121-Address
Identifies points on the public data network at which the object can be reached or with which it is associated in some other important way. The matching rules require that differences caused by the presence of spaces are considered insignificant.

7.3 Selected Object Classes

This Section presents the object classes that are defined in the standards. Object classes are groups of directory entries which share certain characteristics. The object classes are arranged into a lattice, based on the object class **Top**. (In a lattice, each element, except a leaf, has one or more immediate subordinates but also has one or more immediate superiors. This contrasts with a tree, where each element has exactly one immediate superior.) Object classes closer to **Top** are called superclasses, and those further away are called subclasses. This relationship is not connected to any other such relationship in this specification.

Each directory entry belongs to an object class, and to all the superclasses of that object class. Each entry has an attribute named *Object Class*, which was discussed in the previous section, and which identifies the object classes to which the entry belongs. The values of this attribute are object-identifiers, which are represented in the interface by constants with the same name as the object class, prefixed by O- for ease of reading. (Consequently, the C constants are prefixed by DS_O_.)

Associated with each object class are zero or more mandatory and zero or more optional attributes. Each directory entry must contain all the mandatory attributes and may (but need not) contain the optional attributes associated with the object class and its superclasses.

The object classes defined in the standards are tabulated below, together with their object identifiers.

Package	Object Class	Object Identifier decimal	BER hex
BDCP	**O-Alias**	85, 6, 1	\x55\x06\x01
BDCP	**O-Application-Entity**	85, 6, 12	\x55\x06\x0C
BDCP	**O-Application-Process**	85, 6, 11	\x55\x06\x0B
SAP	**O-Cert-Authority**	85, 6, 16	\x55\x06\x10
BDCP	**O-Country**	85, 6, 2	\x55\x06\x02
BDCP	**O-Device**	85, 6, 14	\x55\x06\x0E
BDCP	**O-DSA**	85, 6, 13	\x55\x06\x0D
BDCP	**O-Group-Of-Names**	85, 6, 9	\x55\x06\x09
BDCP	**O-Locality**	85, 6, 3	\x55\x06\x03
MDUP	**O-MHS-Distribution-List**	86, 5, 1, 0	\x56\x05\x01\x00
MDUP	**O-MHS-Message-Store**	86, 5, 1, 1	\x56\x05\x01\x01
MDUP	**O-MHS-Message-Transfer-Agent**	86, 5, 1, 2	\x56\x05\x01\x02
MDUP	**O-MHS-User**	86, 5, 1, 3	\x56\x05\x01\x03
MDUP	**O-MHS-User-Agent**	86, 5, 1, 4	\x56\x05\x01\x04
BDCP	**O-Organization**	85, 6, 4	\x55\x06\x04
BDCP	**O-Organizational-Person**	85, 6, 7	\x55\x06\x07
BDCP	**O-Organizational-Role**	85, 6, 8	\x55\x06\x08
BDCP	**O-Organizational-Unit**	85, 6, 5	\x55\x06\x05
BDCP	**O-Person**	85, 6, 6	\x55\x06\x06
BDCP	**O-Residential-Person**	85, 6, 10	\x55\x06\x0A
SAP	**O-Strong-Authentication-User**	85, 6, 15	\x55\x06\x0F
BDCP	**O-Top**	85, 6, 0	\x55\x06\x00

The third and fourth columns of this table contain the contents octets of the BER encoding of the object identifier. All BDCP and SAP package object identifiers stem from the root {joint-iso-ccitt ds(5) objectClass(6)}. MDUP object identifiers stem from the root {joint-iso-ccitt mhs-motis(6) arch(5) oc(1).

Table 7-3 Object Identifiers for Selected Object Classes

7.4 OM Class Hierarchy

The remainder of this Chapter defines the additional OM classes used to represent values of the selected attributes described in Section 7.2 on page 97. Some of the selected attributes are represented by OM classes that are used in the interface itself, and hence are defined in Chapter 5, (for example, *Name*). As mentioned in Section 7.1 on page 95, an explanation of the purpose of these attributes is outside the scope of this specification.

This Section depicts the hierarchical organisation of the OM classes that are defined in the following sections, and thus shows which OM classes inherit additional OM attributes from their OM superclasses. Subclassification is indicated by indentation, and the names of abstract OM classes are rendered in italics. Thus, for example, **Certificate-Pair** is an immediate subclass of the abstract OM class *Object*.

The package to which each OM class belongs is indicated after the name.

Object (defined in the referenced **XOM** Specification).
— **Algorithm-Ident** (SAP)
— **Certificate-Pair** (SAP)
— **Certificates** (SAP)
— **Cross-Certificates** (SAP)
— **Facsimile-Telephone-Number** (BDCP)
— **Forward-Certification-Path** (SAP)
— *OR-Address* (MDUOP)[1]
  — **OR-Name** (MDUP)[1]
— **Postal-Address** (BDCP)
— **Search-Criterion** (BDCP)
— **Search-Guide** (BDCP)
— *Signature* (SAP)
  — **Certificate** (SAP)
  — **Certificate-List** (SAP)
  — **Certificate-Sublist** (SAP)
— **DL-Submit-Permission** (MDUP)
— **Teletex-Term-Ident** (BDCP)
— **Telex-Number** (BDCP)

[1] as defined in the referenced **X.400** specification.

This specification does not mandate that any OM classes are encodable using *OM-Encode*( ) and *OM-Decode*( ).

7.5 Algorithm-Identifier

An instance of OM class **Algorithm-Ident** records the encryption algorithm that an object uses to digitally sign messages, together with the parameters of the algorithm.

An instance of this OM class has the attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Algorithm	String(Object-Identifier)	-	1	-
Algorithm-Parameters	any	-	0-1	-

Table 7-4 OM Attributes of an Algorithm-Identifier

Algorithm
An object identifier which uniquely identifies the algorithm used by some object.

Algorithm-Parameters
The values of the algorithm's parameters that are used by the object. The syntax of the parameters is determined by each individual algorithm.

7.6 Certificate

An instance of OM class **Certificate** comprises a user's distinguished name, public key and additional information, all of which is digitally signed by the issuing Certification Authority in order to make the certificate unforgeable. The OM attributes associated with **Signature** (a superclass of **Certificate**) shall be present.

An instance of this OM class has the attributes of its superclasses (*Object*, *Signature*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Serial-Number	Integer	-	1	-
Subject	Object(Name)	-	1	-
Subject-Algorithm	Object(Algorithm-Ident)	-	1	-
Subject-Public-Key	String(Bit)	-	1	-
Validity-Not-After	String(UTC-Time)	0-17	1	-
Validity-Not-Before	String(UTC-Time)	0-17	1	-
Version	Enum(Version)	-	1	v1988

Table 7-5 OM Attributes of a Certificate

Serial-Number
Distinguishes the certificate from all other certificates that were ever or will be issued by the Certification Authority which issued this certificate.

Subject
The subject's name.

Subject-Algorithm
The algorithm which is used by the subject for encryption, and which is associated with the public key.

Subject-Public-Key
The subject's public key, associated with the algorithm.

Version
Identifies the certificate's design. Its value is **v1988**, meaning the design specified in the 1988 version of the standards.

Validity-Not-After
The last day on which the certificate is valid.

Validity-Not-Before
The first day on which the certificate is valid.

7.7 Certificate-List

An instance of OM class **Cert-List** documents the revocation of zero or more certificates. The documentation is provided by the object, which shall be a Certification Authority, and whose signature is affixed to the instance.

An instance of this OM class has the OM attributes of its superclasses (*Object*, *Signature*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Last-Update	String(UTC-Time)	0-17	1	-
Revoked-Certs	Object(Certificate-Sublist)	-	0 or more	-

Table 7-6 OM Attributes of a Certificate-List

Last-Update
The time at which the revocation list was updated to its current state.

Revoked-Certs
Identifies the revoked certificates.

7.8 Certificate-Pair

An instance of OM class **Cert-Pair** contains one or both of a forward and reverse certificate, to assist users in building a certification path.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Forward	Object(Certificate)	-	0-1[1]	-
Reverse	Object(Certificate)	-	0-1[1]	-

[1]At least one of these OM attributes must be present.

Table 7-7 OM Attributes of a Certificate-Pair

Forward
The certificate of a first CA (Certification Authority) issued by a second CA.

Reverse
The certificate of the second CA issued by the first CA.

7.9 Certificate-Sublist

An instance of OM class **Cert-Sublist** documents the revocation of zero or more certificates issued by the Certification Authority whose signature is affixed to the instance.

An instance of this OM class has the OM attributes of its superclasses (*Object*, *Signature*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Revoc-Date	String(UTC-Time)	0-17	0 or more[1]	-
Serial-Numbers	Integer	-	0 or more[1]	-

[1]The values of these two OM attributes parallel one another and shall be equal in number.

Table 7-8 OM Attributes of a Cert-Sublist

Revocation-Date
The epoch at which each of the certificates was revoked. The serial numbers of the certificates are the corresponding values of the OM attribute **Serial-Numbers**.

Serial-Numbers
The serial numbers assigned to the revoked certificates.

7.10 Certificates

An instance of OM class **Cert** contains an authentication Certificate and may contain a certification path.

An instance of this OM class has the attributes of its superclass (Object) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Cert	Object(Certificate)	-	1	-
Cert-Path	Object(Forward-Certification-Path)	-	0-1	-

Table 7-9 OM Attributes of a Certificate

Certificate
The certificate.

Certification-Path
The definition of an object to assist users in building a certification path.

7.11 Cross-Certificates

An instance of OM class **Cross-Cert** contains a set of certificates which may be used to build a certification path.

An instance of this OM class has the attributes of its superclass (Object) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Cert	Object(Certificate)	-	0 or more	-

Table 7-10 OM Attributes of Cross-Certificates

Certificate
The certificate.

7.12 Facsimile-Telephone-Number

An instance of OM class **Facsimile-Telephone-Number** identifies and optionally describes a facsimile terminal.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Parameters	Object(G3-Fax-NBPs)[1]	-	0-1	-
Telephone-Number	String(Printable)[2]	1-32	1	-

[1]As defined in the referenced **X.400** specification.
[2]As permitted by E.123 (for example, +44 582 10101).

Table 7-11 OM Attributes of a Facsimile-Telephone-Number

Parameters
If present, identifies the facsimile terminal's non-basic capabilities.

Telephone-Number
A telephone number by means of which the facsimile terminal is accessed.

7.13 Forward-Certification-Path

An instance of OM class **Fwd-Cert-Path** contains one or more objects of class Cross-Certificates.

An instance of this OM class has the attributes of its superclass (Object) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Cross-Cert	Object(Cross-Certificates)	-	1 or more	-

Table 7-12 OM Attributes of a Forward-Certification-Path

Cross-Certificates
The set of certificates which may be used to build a certification path.

7.14 DL-Submit-Permission

An instance of OM class **DL-Submit-Permission** characterizes an attribute each of whose value is a submit permission. An instance of this OM class has the OM attributes of its superclass - Object and additionally the OM attributes listed below.

	Value	**Value**	**Value**	**Value**
OM Attribute	**Syntax**	**Length**	**Number**	**Initially**
Permission-Type	Enum(Permission-Type)	-	1	-
Individual	Object(OR-Name[1])	-	0 or 1	-
Member-of-DL	Object(OR-Name[1])	-	0 or 1	-
Pattern-Match	Object(OR-Name[1])	-	0 or 1	-
Member-of-Group	Object(Name)	-	0 or more	-

[1] As defined in the referenced **X.400** specification.

Table 7-13 OM Attributes of DL-Submit-Permission

Permission-Type
The type of the permission specified herein. Its value can be one of individual, member-of-dl, pattern-match, or member-of-group.

Individual
The user or (unexpanded) DL any of whose O/R names is equal to the specified O/R Name.

Member-of-DL
Each member of the DL, any of whose O/R names is equal to the specified O/R name, or of each nested DL, recursively.

Pattern-Match
Each user or (unexpanded) DL any of whose O/R names matches the specified O/R name pattern.

Member-of-Group
Each member of the group-of-names whose name is specified, or of each nested group-of-names, recursively.

Note that exactly one of the four name attributes shall be present at any time, according to the value of the Permission-Type attribute.

7.15 Postal-Address

An instance of OM class **Postal-Address** is a postal address.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Postal-Address	String(Directory)	1-30	1-6	-

Table 7-14 OM Attributes of a Postal-Address

Postal-Address
Each value of this OM attribute is one line of the postal address. It typically includes a name, street address, city name, state or province name, postal code and possibly country name.

7.16 Search-Criterion

An instance of OM class **Search-Criterion** is a component of a **Search-Guide** OM object.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attribute-Type	String(Object-Identifier)	-	0-1	-
Criteria	Object(Search-Criterion)	-	0 or more	-
Filter-Item-Type	Enum(Filter-Item-Type)	-	0-1	-
Filter-Type	Enum(Filter-Type)	-	1	item

Table 7-15 OM Attributes of a Search-Criterion

A **Search-Criterion** suggests how to build part of a filter for use in searching the directory. Its meaning depends on the value of its OM attribute **Filter-Type**. If the value is **item**, then the criterion is a suggestion for building an instance of OM class **Filter-Item**; while if **Filter-Type** has any other value, it is a suggestion for building an instance of OM class **Filter**.

Attribute-Type
The attribute type to be used in the suggested **Filter-Item**. This OM attribute is only present when the value of **Filter-Type** is **item**.

Criteria
Nested search criteria. This OM attribute is not present when the value of **Filter-Type** is **item**.

Filter-Item-Type
The type of the suggested filter item. Its value can be one of **approximate-match**, **equality**, **greater-or-equal**, **less-or-equal** or **substrings** but not **present**. This OM attribute is only present when the value of **Filter-Type** is **item**.

Filter-Type
The type of the suggested filter. The value **item** means that the suggested component is a filter item, not a filter. The other values suggest the corresponding type of filter. Its value is one of **and**, **item**, **not**, **or**.

7.17 Search-Guide

An instance of OM class **Search-Guide** suggests a criterion for searching the directory for particular entries. It can be used to build a **Filter** for *Search*( ) operations that are based on the object in whose entry the search guide occurs.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Object-Class	String(Object-Identifier)	-	0-1	-
Criteria	Object(Search-Criterion)	-	1	-

Table 7-16 OM Attributes of a Search-Guide

Object-Class
Identifies the object class of the entries to which the search guide applies. If this OM attribute is absent, the search guide applies to objects of any class.

Criteria
The suggested search criteria.

7.18 Signature

The abstract OM class *Signature* contains the algorithm identifier used to produce a digital signature and the name of the object that produced it. The scope of the signature is any instance of any subclass of this OM class.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Issuer	Object(Name)	-	1	-
Signature	Object(Algorithm-Ident)	-	1	-
Signature-Value	String(Octet)	-	1	-

Table 7-17 OM Attributes of a Signature

Issuer
The name of the object which produced the digital signature.

Signature
Identifies the algorithm that was used to produce the digital signature, and any parameters of the algorithm.

Signature-Value
An enciphered summary of the information to which the signature is appended. The summary is produced by means of a one-way hash function, while the enciphering is carried out using the secret key of the signer.

7.19 Teletex-Terminal-Identifier

An instance of OM class **Teletex-Term-Ident** identifies and describes a teletex terminal.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Parameters	Object(Teletex-NBPs)[1]	-	0-1	-
Teletex-Terminal	String(Printable)[2]	1-1024	1	-

[1]As defined in the referenced **X.400** specification.
[2]As permitted by F.200.

Table 7-18 OM Attributes of a Teletex-Term-Ident

Parameters
Identifies the teletex terminal's non-basic capabilities.

Teletex-Terminal
Identifies the teletex terminal.

7.20 Telex-Number

An instance of OM class **Telex-Number** identifies and describes a telex terminal.

An instance of this OM class has the OM attributes of its superclass (*Object*) and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Answerback	String(Printable)	1-8	1	-
Country-Code	String(Printable)	1-4	1	-
Telex-Number	String(Printable)	1-14	1	-

Table 7-19 OM Attributes of a Telex-Number

Answerback
The code with which the telex terminal acknowledges calls placed to it.

Country-Code
The identifier of the country through which the telex terminal is accessed.

Telex-Number
The number by means of which the telex terminal is addressed.

Chapter 8

Headers

8.1 Introduction

Each implementation must provide an **<xds.h>** header for use by applications programs and, if it supports the Basic Directory Contents Package, the Strong Authentication Package or the MHS Directory User Package, must provide an **<xdsbdcp.h>**, an **<xdssap.h>** or an **<xdsmdup.h>** header as appropriate. This chapter sets out the symbols which are defined in these headers. The implementation is not constrained to reproduce the definitions in the exact form set out here, provided that the effects of the definitions contained in this chapter are achieved.

Where the values of the symbols are indicated, the values are an integral part of the interface. Where a value is not given, the value on a particular system will be determined by the vendor or by an administrator.

8.2 <xds.h>

The **<xds.h>** header declares the interface functions, the structures passed to and from those functions, and the defined constants used by the functions and structures.

When an application program includes the **<xds.h>** header, all the definitions made in the Object Management header **<xom.h>** will also apply, as though the application had included **<xom.h>** as well as **<xds.h>**.

```
#ifndef XDS_HEADER
#define XDS_HEADER


/* DS package object identifier */
    define OMP_O_DS_SERVICE_PKG     \x2A\x86\x48\xCE\x1E\x00


/* Typedefs */

typedef OM_private_object  DS_status;
typedef struct {
         OM_object_identifier           feature;
         OM_boolean                     activated;
} DS_feature;
```

```
/* Function Prototypes */

DS_status ds_abandon(
        OM_private_object     session,
        OM_sint               invoke_id
);

DS_status ds_add_entry(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_object             entry,
        OM_sint               *invoke_id_return
);

DS_status ds_bind(
        OM_object             session,
        OM_workspace          workspace,
        OM_private_object     *bound_session_return
);

DS_status ds_compare(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_object             ava,
        OM_private_object     result_return,
        OM_sint               *invoke_id_return
);


OM_workspace ds_initialize(
        void
);

DS_status ds_list(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_private_object     *result_return,
        OM_sint               *invoke_id_return
);

DS_status ds_modify_entry(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_object             changes,
        OM_sint               *invoke_id_return
);
```

```
DS_status ds_modify_rdn(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_object             new_RDN,
        OM_boolean            delete_old_RDN,
        OM_sint               *invoke_id_return
);

DS_status ds_read(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_object             selection,
        OM_private_object     *result_return,
        OM_sint               *invoke_id_return
);

DS_status ds_receive_result(
        OM_private_object     session,
        OM_sint               invoke_id,
        OM_uint               *completion_flag_return,
        DS_status             *operation_status_return,
        OM_private_object     *result_return,
        OM_sint               *invoke_id_return
);


DS_status ds_remove_entry(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_sint               *invoke_id_return
);

DS_status ds_search(
        OM_private_object     session,
        OM_private_object     context,
        OM_object             name,
        OM_sint               subset,
        OM_object             filter,
        OM_boolean            search_aliases,
        OM_object             selection,
        OM_private_object     *result_return,
        OM_sint               *invoke_id_return
);

DS_status ds_shutdown(
        OM_workspace          workspace
);

DS_status ds_unbind(
        OM_private_object     session
);
```

```
DS_status ds_version(
        DS_feature          feature_list[],
        OM_workspace        workspace
);



/* Defined constants                                          */

/* Intermediate object identifier macro                       */
    #define dsP_c(X)     (OMP_O_DS_SERVICE_PKG  #X)

/* OM class names (prefixed DS_C_)                            */

/* Every application program which makes use of a class or    */
/* other Object Identifier must explicitly import it into     */
/* every compilation unit (C source program) which uses it.   */
/* Each such class or Object Identifier name must be          */
/* explicitly exported from just one compilation unit.        */

/* In the header file, OM class constants are prefixed with   */
/* the OPM_O prefix to denote that they are OM classes.       */
/* However, when using the OM_IMPORT and OM_EXPORT macros,    */
/* the base names (without the OMP_O prefix) should be used.  */
/* For example:                                               */
/*     OM_IMPORT(DS_C_AVA)                                    */

   #define OMP_O_DS_C_ABANDON_FAILED         dsP_c(\x85\x3D)
   #define OMP_O_DS_C_ACCESS_POINT           dsP_c(\x85\x3E)
   #define OMP_O_DS_C_ADDRESS                dsP_c(\x85\x3F)
   #define OMP_O_DS_C_ATTRIBUTE              dsP_c(\x85\x40)
   #define OMP_O_DS_C_ATTRIBUTE_ERROR        dsP_c(\x85\x41)
   #define OMP_O_DS_C_ATTRIBUTE_LIST         dsP_c(\x85\x42)
   #define OMP_O_DS_C_ATTRIBUTE_PROBLEM      dsP_c(\x85\x43)
   #define OMP_O_DS_C_AVA                    dsP_c(\x85\x44)
   #define OMP_O_DS_C_COMMON_RESULTS         dsP_c(\x85\x45)
   #define OMP_O_DS_C_COMMUNICATIONS_ERROR   dsP_c(\x85\x46)
   #define OMP_O_DS_C_COMPARE_RESULT         dsP_c(\x85\x47)
   #define OMP_O_DS_C_CONTEXT                dsP_c(\x85\x48)
   #define OMP_O_DS_C_CONTINUATION_REF       dsP_c(\x85\x49)
   #define OMP_O_DS_C_DS_DN                  dsP_c(\x85\x4A)
   #define OMP_O_DS_C_DS_RDN                 dsP_c(\x85\x4B)
   #define OMP_O_DS_C_ENTRY_INFO             dsP_c(\x85\x4C)
   #define OMP_O_DS_C_ENTRY_INFO_SELECTION   dsP_c(\x85\x4D)
   #define OMP_O_DS_C_ENTRY_MOD              dsP_c(\x85\x4E)
   #define OMP_O_DS_C_ENTRY_MOD_LIST         dsP_c(\x85\x4F)
   #define OMP_O_DS_C_ERROR                  dsP_c(\x85\x50)
   #define OMP_O_DS_C_EXT                    dsP_c(\x85\x51)
   #define OMP_O_DS_C_FILTER                 dsP_c(\x85\x52)
   #define OMP_O_DS_C_FILTER_ITEM            dsP_c(\x85\x53)
   #define OMP_O_DS_C_LIBRARY_ERROR          dsP_c(\x85\x54)
   #define OMP_O_DS_C_LIST_INFO              dsP_c(\x85\x55)
   #define OMP_O_DS_C_LIST_INFO_ITEM         dsP_c(\x85\x56)
   #define OMP_O_DS_C_LIST_RESULT            dsP_c(\x85\x57)
   #define OMP_O_DS_C_NAME                   dsP_c(\x85\x58)
   #define OMP_O_DS_C_NAME_ERROR             dsP_c(\x85\x59)
```

```
   #define OMP_O_DS_C_OPERATION_PROGRESS     dsP_c(\x85\x5A)
   #define OMP_O_DS_C_PARTIAL_OUTCOME_QUAL   dsP_c(\x85\x5B)
   #define OMP_O_DS_C_PRESENTATION_ADDRESS   dsP_c(\x85\x5C)
   #define OMP_O_DS_C_READ_RESULT            dsP_c(\x85\x5D)
   #define OMP_O_DS_C_REFERRAL               dsP_c(\x85\x5E)
   #define OMP_O_DS_C_RELATIVE_NAME          dsP_c(\x85\x5F)
   #define OMP_O_DS_C_SEARCH_INFO            dsP_c(\x85\x60)
   #define OMP_O_DS_C_SEARCH_RESULT          dsP_c(\x85\x61)
   #define OMP_O_DS_C_SECURITY_ERROR         dsP_c(\x85\x62)
   #define OMP_O_DS_C_SERVICE_ERROR          dsP_c(\x85\x63)
   #define OMP_O_DS_C_SESSION                dsP_c(\x85\x64)
   #define OMP_O_DS_C_SYSTEM_ERROR           dsP_c(\x85\x65)
   #define OMP_O_DS_C_UPDATE_ERROR           dsP_c(\x85\x66)


/* OM attribute names */

   #define DS_ACCESS_POINTS              ((OM_type) 701)
   #define DS_ADDRESS                    ((OM_type) 702)
   #define DS_AE_TITLE                   ((OM_type) 703)
   #define DS_ALIASED_RDNS               ((OM_type) 704)
   #define DS_ALIAS_DEREFERENCED         ((OM_type) 705)
   #define DS_ALIAS_ENTRY                ((OM_type) 706)
   #define DS_ALL_ATTRIBUTES             ((OM_type) 707)
   #define DS_ASYNCHRONOUS               ((OM_type) 708)
   #define DS_ATTRIBUTES                 ((OM_type) 709)
   #define DS_ATTRIBUTES_SELECTED        ((OM_type) 710)
   #define DS_ATTRIBUTE_TYPE             ((OM_type) 711)
   #define DS_ATTRIBUTE_VALUE            ((OM_type) 712)
   #define DS_ATTRIBUTE_VALUES           ((OM_type) 713)
   #define DS_AUTOMATIC_CONTINUATION     ((OM_type) 714)
   #define DS_AVAS                       ((OM_type) 715)
   #define DS_CHAINING_PROHIB            ((OM_type) 716)
   #define DS_CHANGES                    ((OM_type) 717)
   #define DS_CRIT                       ((OM_type) 718)
   #define DS_DONT_DEREFERENCE_ALIASES   ((OM_type) 719)
   #define DS_DONT_USE_COPY              ((OM_type) 720)
   #define DS_DSA_ADDRESS                ((OM_type) 721)
   #define DS_DSA_NAME                   ((OM_type) 722)
   #define DS_ENTRIES                    ((OM_type) 723)
   #define DS_ENTRY                      ((OM_type) 724)
   #define DS_EXT                        ((OM_type) 725)
   #define DS_FILE_DESCRIPTOR            ((OM_type) 726)
   #define DS_FILTERS                    ((OM_type) 727)
   #define DS_FILTER_ITEMS               ((OM_type) 728)
   #define DS_FILTER_ITEM_TYPE           ((OM_type) 729)
   #define DS_FILTER_TYPE                ((OM_type) 730)
   #define DS_FINAL_SUBSTRING            ((OM_type) 731)
   #define DS_FROM_ENTRY                 ((OM_type) 732)
   #define DS_IDENT                      ((OM_type) 733)
   #define DS_INFO_TYPE                  ((OM_type) 734)
   #define DS_INITIAL_SUBSTRING          ((OM_type) 735)
   #define DS_ITEM_PARAMETERS            ((OM_type) 736)
   #define DS_LIMIT_PROBLEM              ((OM_type) 737)
```

```
#define DS_LIST_INFO                ((OM_type) 738)
#define DS_LOCAL_SCOPE              ((OM_type) 739)
#define DS_MATCHED                  ((OM_type) 740)
#define DS_MOD_TYPE                 ((OM_type) 741)
#define DS_NAME_RESOLUTION_PHASE    ((OM_type) 742)
#define DS_NEXT_RDN_TO_BE_RESOLVED  ((OM_type) 743)
#define DS_N_ADDRESSES              ((OM_type) 744)
#define DS_OBJECT_NAME              ((OM_type) 745)
#define DS_OPERATION_PROGRESS       ((OM_type) 746)
#define DS_PARTIAL_OUTCOME_QUAL     ((OM_type) 747)
#define DS_PERFORMER                ((OM_type) 748)
#define DS_PREFER_CHAINING          ((OM_type) 749)
#define DS_PRIORITY                 ((OM_type) 750)
#define DS_PROBLEM                  ((OM_type) 751)
#define DS_PROBLEMS                 ((OM_type) 752)
#define DS_P_SELECTOR               ((OM_type) 753)
#define DS_RDN                      ((OM_type) 754)
#define DS_RDNS                     ((OM_type) 755)
#define DS_RDNS_RESOLVED            ((OM_type) 756)
#define DS_REQUESTOR                ((OM_type) 757)
#define DS_SCOPE_OF_REFERRAL        ((OM_type) 758)
#define DS_SEARCH_INFO              ((OM_type) 759)
#define DS_SIZE_LIMIT               ((OM_type) 760)
#define DS_SUBORDINATES             ((OM_type) 761)
#define DS_S_SELECTOR               ((OM_type) 762)
#define DS_TARGET_OBJECT            ((OM_type) 763)
#define DS_TIME_LIMIT               ((OM_type) 764)
#define DS_T_SELECTOR               ((OM_type) 765)
#define DS_UNAVAILABLE_CRIT_EXT     ((OM_type) 766)
#define DS_UNCORRELATED_LIST_INFO   ((OM_type) 767)
#define DS_UNCORRELATED_SEARCH_INFO ((OM_type) 768)
#define DS_UNEXPLORED               ((OM_type) 769)


/* DS_Filter_Item_Type */

enum DS_Filter_Item_Type {
     DS_EQUALITY            =  0
     DS_SUBSTRINGS          =  1
     DS_GREATER_OR_EQUAL    =  2
     DS_LESS_OR_EQUAL       =  3
     DS_PRESENT             =  4
     DS_APPROXIMATE_MATCH   =  5
};


/* DS_Filter_Type */

enum DS_Filter_Type {
     DS_ITEM          =  0
     DS_AND           =  1
     DS_OR            =  2
     DS_NOT           =  3
```

```
   };


/* DS_Information_Type */

   enum DS_Information_Type {
        DS_TYPES_ONLY           =  0
        DS_TYPES_AND_VALUES     =  1
   };


/* DS_Limit_Problem */

   enum DS_Limit_Problem {
        DS_TIME_LIMIT_EXCEEDED    =  0
        DS_SIZE_LIMIT_EXCEEDED    =  1
        DS_ADMIN_LIMIT_EXCEEDED   =  2
   };


/* DS_Modification_Type */

   enum DS_Modification_Type {
        DS_ADD_ATTRIBUTE         =  0
        DS_REMOVE_ATTRIBUTE      =  1
        DS_ADD_VALUES            =  2
        DS_REMOVE_VALUES         =  3
   };


/* DS_Name_Resolution_Phase */

   enum DS_Name_Resolution_Phase {
        DS_NOT_STARTED               =  1
        DS_PROCEEDING                =  2
        DS_COMPLETED                 =  3
   };


/* DS_Priority */

   enum DS_Priority {
        DS_LOW          =  0
        DS_MEDIUM       =  1
        DS_HIGH         =  2
   };
```

```
/* DS_Problem */

   enum DS_Problem {
        DS_E_ADMIN_LIMIT_EXCEEDED          =  1
        DS_E_AFFECTS_MULTIPLE_DSAS         =  2
        DS_E_ALIAS_DEREFERENCING_PROBLEM   =  3
        DS_E_ALIAS_PROBLEM                 =  4
        DS_E_ATTRIBUTE_OR_VALUE_EXISTS     =  5
        DS_E_BAD_ARGUMENT                  =  6
        DS_E_BAD_CLASS                     =  7
        DS_E_BAD_CONTEXT                   =  8
        DS_E_BAD_NAME                      =  9
        DS_E_BAD_SESSION                   = 10
        DS_E_BAD_WORKSPACE                 = 11
        DS_E_BUSY                          = 12
        DS_E_CANNOT_ABANDON                = 13
        DS_E_CHAINING_REQUIRED             = 14
        DS_E_COMMUNICATIONS_PROBLEM        = 15
        DS_E_CONSTRAINT_VIOLATION          = 16
        DS_E_DIT_ERROR                     = 17
        DS_E_ENTRY_EXISTS                  = 18
        DS_E_INAPPROP_AUTHENTICATION       = 19
        DS_E_INAPPROP_MATCHING             = 20
        DS_E_INSUFFICIENT_ACCESS_RIGHTS    = 21
        DS_E_INVALID_ATTRIBUTE_SYNTAX      = 22
        DS_E_INVALID_ATTRIBUTE_VALUE       = 23
        DS_E_INVALID_CREDENTIALS           = 24
        DS_E_INVALID_REF                   = 25
        DS_E_INVALID_SIGNATURE             = 26
        DS_E_LOOP_DETECTED                 = 27
        DS_E_MISCELLANEOUS                 = 28
        DS_E_MISSING_TYPE                  = 29
        DS_E_MIXED_SYNCHRONOUS             = 30
        DS_E_NAMING_VIOLATION              = 31
        DS_E_NO_INFO                       = 32
        DS_E_NO_SUCH_ATTRIBUTE_OR_VALUE    = 33
        DS_E_NO_SUCH_OBJECT                = 34
        DS_E_NO_SUCH_OPERATION             = 35
        DS_E_NOT_ALLOWED_ON_NON_LEAF       = 36
        DS_E_NOT_ALLOWED_ON_RDN            = 37
        DS_E_NOT_SUPPORTED                 = 38
        DS_E_OBJECT_CLASS_MOD_PROHIB       = 39
        DS_E_OBJECT_CLASS_VIOLATION        = 40
        DS_E_OUT_OF_SCOPE                  = 41
        DS_E_PROTECTION_REQUIRED           = 42
        DS_E_TIME_LIMIT_EXCEEDED           = 43
        DS_E_TOO_LATE                      = 44
        DS_E_TOO_MANY_OPERATIONS           = 45
        DS_E_TOO_MANY_SESSIONS             = 46
        DS_E_UNABLE_TO_PROCEED             = 47
        DS_E_UNAVAILABLE                   = 48
        DS_E_UNAVAILABLE_CRIT_EXT          = 49
        DS_E_UNDEFINED_ATTRIBUTE_TYPE      = 50
        DS_E_UNWILLING_TO_PERFORM          = 51
```

```
   };


/* DS_Scope_Of_Referral */

   enum DS_Scope_Of_Referral {
        DS_DMD                  =  0
        DS_COUNTRY              =  1
   };


/* OM_object constants */

   #define DS_DEFAULT_CONTEXT               ((OM_object) 0)
   #define DS_DEFAULT_SESSION               ((OM_object) 0)
   #define DS_OPERATION_NOT_STARTED         ((OM_object) 0)
   #define DS_NO_FILTER                     ((OM_object) 0)
   #define DS_NULL_RESULT                   ((OM_object) 0)
   #define DS_SELECT_ALL_TYPES              ((OM_object) 1)
   #define DS_SELECT_ALL_TYPES_AND_VALUES   ((OM_object) 2)
   #define DS_SELECT_NO_ATTRIBUTES          ((OM_object) 0)
   #define DS_SUCCESS                       ((DS_status) 0)
   #define DS_INVALID_WORKSPACE             ((DS_status) 2)
   #define DS_NO_WORKSPACE                  ((DS_status) 1)


/* ds_search subset */

   #define DS_BASE_OBJECT     ( (OM_sint) 0)
   #define DS_ONE_LEVEL       ( (OM_sint) 1)
   #define DS_WHOLE_SUBTREE   ( (OM_sint) 2)


/* ds_receive_result completion_flag_return */

   #define DS_COMPLETED_OPERATION          ( (OM_uint) 1)
   #define DS_OUTSTANDING_OPERATIONS       ( (OM_uint) 2)
   #define DS_NO_OUTSTANDING_OPERATION     ( (OM_uint) 3)
   #define DS_OTHER_COMPLETED_OPERATIONS   ( (OM_uint) 4)


/* Invocation Identifier Flags for ds_receive_result */

   #define DS_ANY_OPERATION   ( (OM_sint) 0)


/* asynchronous operations limit (implementation-defined) */

   #define DS_MAX_OUTSTANDING_OPERATIONS   1


/* asynchronous event posting */
```

```
#define DS_NO_VALID_FILE_DESCRIPTOR   -1

#endif  /* XDS_HEADER */
```

8.3 <xdsbdcp.h>

The **<xdsbdcp.h>** header defines the object identifiers of directory attribute types and object classes supported by the Basic Directory Contents Package. It also defines OM classes used to represent the values of the attribute types.

When an application program includes the **<xdsbdcp.h>** header, all the definitions made in the Object Management header **<xom.h>** and the **<xds.h>** header will also apply, as though the application had included **<xom.h>** and **<xds.h>** as well as **<xdsbdcp.h>**.

```
#ifndef XDSBDCP_HEADER
#define XDSBDCP_HEADER

#define OMP_O_DS_BASIC_DIR_CONTENTS_PKG   \x2A\x86\x48\xCE\x1E\x01

/* Intermediate object identifier macros */

#ifndef dsP_attributeType
    #define dsP_attributeType(X)   ("\x55\x4" #X) joint-iso-ccitt 5 4
#endif

#ifndef dsP_objectClass
      #define dsP_objectClass(X)   ("\x55\x6" #X) joint-iso-ccitt 5 6
#endif


#define dsP_bdcp_c(X)   (OMP_O_DS_BASIC_DIR_CONTENTS_PKG  #X)


/* OM class names (prefixed DS_C_),                          */
/* Directory attribute types (prefixed DS_A_),               */
/* and Directory object classes (prefixed DS_O_)             */

/* Every application program which makes use of a class or   */
/* other Object Identifier must explicitly import it into    */
/* every compilation unit (C source program) which uses it.  */
/* Each such class or Object Identifier name must be         */
/* explicitly exported from just one compilation unit.       */

/* In the header file, OM class constants are prefixed with  */
/* the OPM_O prefix to denote that they are OM classes.      */
/* However, when using the OM_IMPORT and OM_EXPORT macros,   */
/* the base names (without the OMP_O prefix) should be used. */
/* For example:                                              */
/*     OM_IMPORT(DS_O_COUNTRY)                               */


/* Directory attribute types */

   #define OMP_O_DS_A_ALIASED_OBJECT_NAME      dsP_attributeType(\x01)
   #define OMP_O_DS_A_BUSINESS_CATEGORY        dsP_attributeType(\x0F)
   #define OMP_O_DS_A_COMMON_NAME              dsP_attributeType(\x03)
   #define OMP_O_DS_A_COUNTRY_NAME             dsP_attributeType(\x06)
   #define OMP_O_DS_A_DESCRIPTION              dsP_attributeType(\x0D)
   #define OMP_O_DS_A_DEST_INDICATOR           dsP_attributeType(\x1B)
   #define OMP_O_DS_A_FACSIMILE_PHONE_NBR      dsP_attributeType(\x17)
```

```
   #define OMP_O_DS_A_INTERNAT_ISDN_NBR        dsP_attributeType(\x19)
   #define OMP_O_DS_A_KNOWLEDGE_INFO           dsP_attributeType(\x02)
   #define OMP_O_DS_A_LOCALITY_NAME            dsP_attributeType(\x07)
   #define OMP_O_DS_A_MEMBER                   dsP_attributeType(\x1F)
   #define OMP_O_DS_A_OBJECT_CLASS             dsP_attributeType(\x00)
   #define OMP_O_DS_A_ORG_NAME                 dsP_attributeType(\x0A)
   #define OMP_O_DS_A_ORG_UNIT_NAME            dsP_attributeType(\x0B)
   #define OMP_O_DS_A_OWNER                    dsP_attributeType(\x20)
   #define OMP_O_DS_A_PHONE_NBR                dsP_attributeType(\x14)
   #define OMP_O_DS_A_PHYS_DELIV_OFF_NAME      dsP_attributeType(\x13)
   #define OMP_O_DS_A_POST_OFFICE_BOX          dsP_attributeType(\x12)
   #define OMP_O_DS_A_POSTAL_ADDRESS           dsP_attributeType(\x10)
   #define OMP_O_DS_A_POSTAL_CODE              dsP_attributeType(\x11)
   #define OMP_O_DS_A_PREF_DELIV_METHOD        dsP_attributeType(\x1C)
   #define OMP_O_DS_A_PRESENTATION_ADDRESS     dsP_attributeType(\x1D)
   #define OMP_O_DS_A_REGISTERED_ADDRESS       dsP_attributeType(\x1A)
   #define OMP_O_DS_A_ROLE_OCCUPANT            dsP_attributeType(\x21)
   #define OMP_O_DS_A_SEARCH_GUIDE             dsP_attributeType(\x0E)
   #define OMP_O_DS_A_SEE_ALSO                 dsP_attributeType(\x22)
   #define OMP_O_DS_A_SERIAL_NBR               dsP_attributeType(\x05)
   #define OMP_O_DS_A_STATE_OR_PROV_NAME       dsP_attributeType(\x08)
   #define OMP_O_DS_A_STREET_ADDRESS           dsP_attributeType(\x09)
   #define OMP_O_DS_A_SUPPORT_APPLIC_CONTEXT   dsP_attributeType(\x1E)
   #define OMP_O_DS_A_SURNAME                  dsP_attributeType(\x04)
   #define OMP_O_DS_A_TELETEX_TERM_IDENT       dsP_attributeType(\x16)
   #define OMP_O_DS_A_TELEX_NBR                dsP_attributeType(\x15)
   #define OMP_O_DS_A_TITLE                    dsP_attributeType(\x0C)
   #define OMP_O_DS_A_USER_PASSWORD            dsP_attributeType(\x23)
   #define OMP_O_DS_A_X121_ADDRESS             dsP_attributeType(\x18)


/* Directory object classes */

   #define OMP_O_DS_O_ALIAS                dsP_objectClass(\x01)
   #define OMP_O_DS_O_APPLIC_ENTITY        dsP_objectClass(\x0C)
   #define OMP_O_DS_O_APPLIC_PROCESS       dsP_objectClass(\x0B)
   #define OMP_O_DS_O_COUNTRY              dsP_objectClass(\x02)
   #define OMP_O_DS_O_DEVICE               dsP_objectClass(\x0E)
   #define OMP_O_DS_O_DSA                  dsP_objectClass(\x0D)
   #define OMP_O_DS_O_GROUP_OF_NAMES       dsP_objectClass(\x09)
   #define OMP_O_DS_O_LOCALITY             dsP_objectClass(\x03)
   #define OMP_O_DS_O_ORG                  dsP_objectClass(\x04)
   #define OMP_O_DS_O_ORG_PERSON           dsP_objectClass(\x07)
   #define OMP_O_DS_O_ORG_ROLE             dsP_objectClass(\x08)
   #define OMP_O_DS_O_ORG_UNIT             dsP_objectClass(\x05)
   #define OMP_O_DS_O_PERSON               dsP_objectClass(\x06)
   #define OMP_O_DS_O_RESIDENTIAL_PERSON   dsP_objectClass(\x0A)
   #define OMP_O_DS_O_TOP                  dsP_objectClass(\x00)
```

```
/* OM class names */

   #define OMP_O_DS_C_FACSIMILE_PHONE_NBR   dsP_bdcp_c(\x86\x21)
   #define OMP_O_DS_C_POSTAL_ADDRESS        dsP_bdcp_c(\x86\x22)
   #define OMP_O_DS_C_SEARCH_CRITERION      dsP_bdcp_c(\x86\x23)
   #define OMP_O_DS_C_SEARCH_GUIDE          dsP_bdcp_c(\x86\x24)
   #define OMP_O_DS_C_TELETEX_TERM_IDENT    dsP_bdcp_c(\x86\x25)
   #define OMP_O_DS_C_TELEX_NBR             dsP_bdcp_c(\x86\x26)


/* OM attribute names */

   #define DS_ANSWERBACK       ((OM_type) 801)
   #define DS_COUNTRY_CODE     ((OM_type) 802)
   #define DS_CRITERIA         ((OM_type) 803)
   #define DS_OBJECT_CLASS     ((OM_type) 804)
   #define DS_PARAMETERS       ((OM_type) 805)
   #define DS_POSTAL_ADDRESS   ((OM_type) 806)
   #define DS_PHONE_NBR        ((OM_type) 807)
   #define DS_TELETEX_TERM     ((OM_type) 808)
   #define DS_TELEX_NBR        ((OM_type) 809)


/* DS_Preferred_Delivery_Method  */

   enum DS_Preferred_Delivery_Method{
        DS_ANY_DELIV_METHOD             = 0
        DS_MHS_DELIV                    = 1
        DS_PHYS_DELIV                   = 2
        DS_TELEX_DELIV                  = 3
        DS_TELETEX_DELIV                = 4
        DS_G3_FACSIMILE_DELIV           = 5
        DS_G4_FACSIMILE_DELIV           = 6
        DS_IA5_TERMINAL_DELIV           = 7
        DS_VIDEOTEX_DELIV               = 8
        DS_PHONE_DELIV                  = 9
   };


/*upper bounds on string lengths and number of repeated OM attribute values*/

   #define DS_VL_A_BUSINESS_CATEGORY     ( (OM_value_length) 128)
   #define DS_VL_A_COMMON_NAME           ( (OM_value_length) 64)
   #define DS_VL_A_DESCRIPTION           ( (OM_value_length) 1024)
   #define DS_VL_A_DEST_INDICATOR        ( (OM_value_length) 128)
   #define DS_VL_A_INTERNAT_ISDN_NBR     ( (OM_value_length) 16)
   #define DS_VL_A_LOCALITY_NAME         ( (OM_value_length) 128)
   #define DS_VL_A_ORG_NAME              ( (OM_value_length) 64)
   #define DS_VL_A_ORG_UNIT_NAME         ( (OM_value_length) 64)
   #define DS_VL_A_PHYS_DELIV_OFF_NAME   ( (OM_value_length) 128)
   #define DS_VL_A_POST_OFFICE_BOX       ( (OM_value_length) 40)
   #define DS_VL_A_POSTAL_CODE           ( (OM_value_length) 40)
```

```
    #define DS_VL_A_SERIAL_NBR           ( (OM_value_length) 64)
    #define DS_VL_A_STATE_OR_PROV_NAME   ( (OM_value_length) 128)
    #define DS_VL_A_STREET_ADDRESS       ( (OM_value_length) 128)
    #define DS_VL_A_SURNAME              ( (OM_value_length) 64)
    #define DS_VL_A_PHONE_NBR            ( (OM_value_length) 32)
    #define DS_VL_A_TITLE                ( (OM_value_length) 64)
    #define DS_VL_A_USER_PASSWORD        ( (OM_value_length) 128)
    #define DS_VL_A_X121_ADDRESS         ( (OM_value_length) 15)
    #define DS_VL_ANSWERBACK             ( (OM_value_length) 8)
    #define DS_VL_COUNTRY_CODE           ( (OM_value_length) 4)
    #define DS_VL_POSTAL_ADDRESS         ( (OM_value_length) 30)
    #define DS_VL_PHONE_NBR              ( (OM_value_length) 32)
    #define DS_VL_TELETEX_TERM           ( (OM_value_length) 1024)
    #define DS_VL_TELEX_NBR              ( (OM_value_length) 14)
    #define DS_VN_POSTAL_ADDRESS         ( (OM_value_length) 6)


#endif  /* XDSBDCP_HEADER */
```

8.4 <xdssap.h>

The **<xdssap.h>** header defines the object identifiers of directory attribute types and object classes supported by the Strong Authentication Package. It also defines OM classes used to represent the values of the attribute types.

When an application program includes the **<xdssap.h>** header, all the definitions made in the Object Management header **<xom.h>** and the **<xds.h>** header will also apply, as though the application had included **<xom.h>** and **<xds.h>** as well as **<xdssap.h>**.

```
#ifndef XDSSAP_HEADER
#define XDSSAP_HEADER

#define OMP_O_DS_STRONG_AUTHENT_PKG    \x2A\x86\x48\xCE\x1E\x02

/* Intermediate object identifier macros */

#ifndef dsP_attributeType
      #define dsP_attributeType(X)   ("\x55\x4" #X)   joint-iso-ccitt 5 4
#endif

#ifndef dsP_objectClass
      #define dsP_objectClass(X)     ("\x55\x6" #X)   joint-iso-ccitt 5 6
#endif


#define dsP_sap_c(X)     (OMP_O_DS_STRONG_AUTHENT_PKG  #X)


/* OM class names (prefixed DS_C_),                          */
/* Directory attribute types (prefixed DS_A_),               */
/* and Directory object classes (prefixed DS_O_)             */

/* Every application program which makes use of a class or   */
/* other Object Identifier must explicitly import it into    */
/* every compilation unit (C source program) which uses it.  */
/* Each such class or Object Identifier name must be         */
/* explicitly exported from just one compilation unit.       */

/* In the header file, OM class constants are prefixed with  */
/* the OPM_O prefix to denote that they are OM classes.      */
/* However, when using the OM_IMPORT and OM_EXPORT macros,   */
/* the base names (without the OMP_O prefix) should be used. */
/* For example:                                              */
/*     OM_IMPORT(DS_O_CERT_AUTHORITY)                        */
```

```
/* Directory attribute types */

   #define OMP_O_DS_A_AUTHORITY_REVOC_LIST   dsP_attributeType(\x26)
   #define OMP_O_DS_A_CA_CERT                dsP_attributeType(\x25)
   #define OMP_O_DS_A_CERT_REVOC_LIST        dsP_attributeType(\x27)
   #define OMP_O_DS_A_CROSS_CERT_PAIR        dsP_attributeType(\x28)
   #define OMP_O_DS_A_USER_CERT              dsP_attributeType(\x24)


/* Directory object classes */

   #define OMP_O_DS_O_CERT_AUTHORITY        dsP_objectClass(\x10)
   #define OMP_O_DS_O_STRONG_AUTHENT_USER   dsP_objectClass(\x0F)


/* OM class names */

   #define OMP_O_DS_C_ALGORITHM_IDENT   dsP_sap_c(\x86\x35)
   #define OMP_O_DS_C_CERT              dsP_sap_c(\x86\x36)
   #define OMP_O_DS_C_CERT_LIST         dsP_sap_c(\x86\x37)
   #define OMP_O_DS_C_CERT_PAIR         dsP_sap_c(\x86\x38)
   #define OMP_O_DS_C_CERT_SUBLIST      dsP_sap_c(\x86\x39)
   #define OMP_O_DS_C_SIGNATURE         dsP_sap_c(\x86\x3A)
   #define OMP_O_DS_C_CERTS             dsP_sap_c(\x86\x3B)
   #define OMP_O_DS_C_CROSS_CERTS       dsP_sap_c(\x86\x3C)
   #define OMP_O_DS_C_FWD_CERT_PATH     dsP_sap_c(\x86\x3D)


/* OM attribute names */

   #define DS_ALGORITHM              ((OM_type) 821)
   #define DS_FORWARD                ((OM_type) 822)
   #define DS_ISSUER                 ((OM_type) 823)
   #define DS_LAST_UPDATE            ((OM_type) 824)
   #define DS_ALGORITHM_PARAMETERS   ((OM_type) 825)
   #define DS_REVERSE                ((OM_type) 826)
   #define DS_REVOC_DATE             ((OM_type) 827)
   #define DS_REVOKED_CERTS          ((OM_type) 828)
   #define DS_SERIAL_NBR             ((OM_type) 829)
   #define DS_SERIAL_NBRS            ((OM_type) 830)
   #define DS_SIGNATURE              ((OM_type) 831)
   #define DS_SIGNATURE_VALUE        ((OM_type) 832)
   #define DS_SUBJECT                ((OM_type) 833)
   #define DS_SUBJECT_ALGORITHM      ((OM_type) 834)
   #define DS_SUBJECT_PUBLIC_KEY     ((OM_type) 835)
   #define DS_VALIDITY_NOT_AFTER     ((OM_type) 836)
   #define DS_VALIDITY_NOT_BEFORE    ((OM_type) 837)
   #define DS_VERSION                ((OM_type) 838)
   #define DS_CERT                   ((OM_type) 839)
   #define DS_CERT_PATH              ((OM_type) 840)
   #define DS_FWD_CERT_PATH          ((OM_type) 841)


/* DS_Version */
```

```
   #define DS_V1988   ( (OM_enumeration) 1)


/*upper bounds on string lengths and number of repeated OM attribute values*/

   #define DS_VL_LAST_UPDATE           ( (OM_value_length) 17)
   #define DS_VL_REVOC_DATE            ( (OM_value_length) 17)
   #define DS_VL_VALIDITY_NOT_AFTER    ( (OM_value_length) 17)
   #define DS_VL_VALIDITY_NOT_BEFORE   ( (OM_value_length) 17)
   #define DS_VN_REVOC_DATE            ( (OM_value_number) 2)


#endif  /* XDSSAP_HEADER */
```

8.5 <xdsmdup.h>

The **<xdsmdup.h>** header defines the object identifiers of directory attribute types and object classes supported by the MHS Directory User Package. It also defines OM classes used to represent the values of the attribute types.

When an application program includes the **<xdsmdup.h>** header, all the definitions made in the Object Management header **<xom.h>** and the **<xds.h>** header will also apply, as though the application had included **<xom.h>** and **<xds.h>** as well as **<xdsmdup.h>**.

```
#ifndef XDSMDUP_HEADER
#define XDSMDUP_HEADER

#ifndef XMHP_HEADER
#include <xmhp.h>
#endif  /* XMHP_HEADER */

/* MDUP package object identifier */
#define OMP_O_DS_MHS_DIR_USER_PKG     \x2A\x86\x48\xCE\x1E\x03

/* Intermediate object identifier macros */

#define dsP_MHSattributeType(X) ("\x56\x5\x2" #X) /* joint-iso-ccitt 6 5 2 */
#define dsP_MHSobjectClass(X)   ("\x56\x5\x1" #X) /* joint-iso-ccitt 6 5 1 */


#define dsP_mdup_c(X)    (OMP_O_DS_MHS_DIR_USER_PKG  #X)


/* OM class names (prefixed DS_C_),                          */
/* Directory attribute types (prefixed DS_A_),               */
/* and Directory object classes (prefixed DS_O_)             */
/* Every application program which makes use of a class or   */
/* other Object Identifier must explicitly import it into    */
/* every compilation unit (C source program) which uses it.  */
/* Each such class or Object Identifier name must be         */
/* explicitly exported from just one compilation unit.       */
/* In the header file, OM class constants are prefixed with  */
/* the OPM_O prefix to denote that they are OM classes.      */
/* However, when using the OM_IMPORT and OM_EXPORT macros,   */
/* the base names (without the OMP_O prefix) should be used. */
/* For example:                                              */
/*     OM_IMPORT(DS_O_CERT_AUTHORITY)                        */
```

```
/* Directory attribute types */

   #define OMP_O_DS_A_DELIV_CONTENT_LENGTH   dsP_MHSattributeType(\x00)
   #define OMP_O_DS_A_DELIV_CONTENT_TYPES    dsP_MHSattributeType(\x01)
   #define OMP_O_DS_A_DELIV_EITS             dsP_MHSattributeType(\x02)
   #define OMP_O_DS_A_DL_MEMBERS             dsP_MHSattributeType(\x03)
   #define OMP_O_DS_A_DL_SUBMIT_PERMS        dsP_MHSattributeType(\x04)
   #define OMP_O_DS_A_MESSAGE_STORE          dsP_MHSattributeType(\x05)
   #define OMP_O_DS_A_OR_ADDRESSES           dsP_MHSattributeType(\x06)
   #define OMP_O_DS_A_PREF_DELIV_METHODS     dsP_MHSattributeType(\x07)
   #define OMP_O_DS_A_SUPP_AUTO_ACTIONS      dsP_MHSattributeType(\x08)
   #define OMP_O_DS_A_SUPP_CONTENT_TYPES     dsP_MHSattributeType(\x09)
   #define OMP_O_DS_A_SUPP_OPT_ATTRIBUTES    dsP_MHSattributeType(\x0A)


/* Directory object classes */

   #define OMP_O_DS_O_MHS_DISTRIBUTION_LIST   dsP_MHSobjectClass(\x00)
   #define OMP_O_DS_O_MHS_MESSAGE_STORE       dsP_MHSobjectClass(\x01)
   #define OMP_O_DS_O_MHS_MESSAGE_TRANS_AG    dsP_MHSobjectClass(\x02)
   #define OMP_O_DS_O_MHS_USER                dsP_MHSobjectClass(\x03)
   #define OMP_O_DS_O_MHS_USER_AG             dsP_MHSobjectClass(\x04)


/* OM class names */

   #define OMP_O_DS_C_DL_SUBMIT_PERMS   dsP_mdup_c(\x87\x05)


/* OM attribute names */

   #define DS_PERM_TYPE         ( (OM_type) 901 )
   #define DS_INDIVIDUAL        ( (OM_type) 902 )
   #define DS_MEMBER_OF_DL      ( (OM_type) 903 )
   #define DS_PATTERN_MATCH     ( (OM_type) 904 )
   #define DS_MEMBER_OF_GROUP   ( (OM_type) 905 )


/* DS_Permission_Type */

   enum DS_Permission_Type {
        DS_PERM_INDIVIDUAL        = 0,
        DS_PERM_MEMBER_OF_DL      = 1,
        DS_PERM_PATTERN_MATCH     = 2,
        DS_PERM_MEMBER_OF_GROUP   = 3
   };


#endif /* XDSMDUP_HEADER */
```

Appendix A

Programming Examples

A.1 Introduction

This Appendix provides two examples of C programs which use this interface to the directory. The first example illustrates the use of synchronous operations, whilst the second makes use of asynchronous operations. Both examples make use of common error handling functions and macros from the **<example.h>** header which is described at the end of this Chapter.

A.2 Synchronous Directory Example

```
/*
 * Sample application that uses XDS in synchronous mode.
 *
 * This program reads the telephone number(s) of a given target name.
 */

#include   <stdio.h>

#include   <xom.h>       /* Object Management header */
#include   <xds.h>       /* Directory Service header */
#include   <xdsbdcp.h>   /* Basic Directory Contents Package header */

#include   "example.h"   /* possible Error Handling header */

/*
 * Define necessary Object Identifier constants.
 */

OM_EXPORT(DS_A_COMMON_NAME)
OM_EXPORT(DS_A_COUNTRY_NAME)
OM_EXPORT(DS_A_ORG_NAME)
OM_EXPORT(DS_A_ORG_UNIT_NAME)
OM_EXPORT(DS_A_PHONE_NBR)
OM_EXPORT(DS_C_AVA)
OM_EXPORT(DS_C_DS_DN)
OM_EXPORT(DS_C_DS_RDN)
OM_EXPORT(DS_C_ENTRY_INFO_SELECTION)

int main(void) {
   DS_status           error;         /* return value from DS functions */
   OM_return_code      return_code;   /* return value from OM functions */
   OM_workspace        workspace;     /* workspace for objects */
   OM_private_object   session;       /* session for directory operations */
   OM_private_object   result;        /* result of read operation */
   OM_sint             invoke_id;     /* Invoke-ID of the read operation */
   OM_value_position   total_number   /* Number of Attribute Descriptors */
};
```

```
static DS_feature bdcp_package[ ] = {
   {DS_BASIC_DIR_CONTENTS_PKG,  OM_TRUE},
   {{(OM_uint32)0,  (void *)0}, OM_FALSE}
   };
/*
 * Public Object ("Descriptor List") for Name argument to ds_read( ).
 * Build the Distinguished-Name of Peter Piper.
 */

static OM_descriptor  country[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_AVA),
   OM_OID_DESC(DS_ATTRIBUTE_TYPE,  DS_A_COUNTRY_NAME),
   { DS_ATTRIBUTE_VALUES,OM_S_PRINTABLE_STRING,{  OM_STRING("US") } },
   OM_NULL_DESCRIPTOR
   };
static OM_descriptor  organization[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_AVA),
   OM_OID_DESC(DS_ATTRIBUTE_TYPE,  DS_A_ORGANIZATION_NAME),
   { DS_ATTRIBUTE_VALUES,OM_S_TELETEX_STRING,{  OM_STRING("Acme Pepper Co") } },
   OM_NULL_DESCRIPTOR
   };
static OM_descriptor  organizational_unit[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_AVA),
   OM_OID_DESC(DS_ATTRIBUTE_TYPE,  DS_A_ORGANIZATIONAL_UNIT_NAME),
   { DS_ATTRIBUTE_VALUES,OM_S_TELETEX_STRING,{  OM_STRING("Research") } },
   OM_NULL_DESCRIPTOR
   };
static OM_descriptor  common_name[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_AVA),
   OM_OID_DESC(DS_ATTRIBUTE_TYPE,  DS_A_COMMON_NAME),
   { DS_ATTRIBUTE_VALUES,OM_S_TELETEX_STRING,{  OM_STRING("Peter Piper") } },
   OM_NULL_DESCRIPTOR
   };
static OM_descriptor  rdn1[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
   { DS_AVAS,  OM_S_OBJECT,  { { 0, country } } },
   OM_NULL_DESCRIPTOR
   };
static OM_descriptor  rdn2[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
   { DS_AVAS,  OM_S_OBJECT,  { { 0, organization } } },
   OM_NULL_DESCRIPTOR
   };
```

```
static OM_descriptor  rdn3[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
   { DS_AVAS,  OM_S_OBJECT,  { { 0, organizational_unit  } } },
   OM_NULL_DESCRIPTOR
   };
static OM_descriptor  rdn4[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
   { DS_AVAS,  OM_S_OBJECT,  { { 0, common_name  } } },
   OM_NULL_DESCRIPTOR
   };
OM_descriptor         name[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_DS_DN),
   { DS_RDNS,  OM_S_OBJECT,  { { 0, rdn1 } } },
   { DS_RDNS,  OM_S_OBJECT,  { { 0, rdn2 } } },
   { DS_RDNS,  OM_S_OBJECT,  { { 0, rdn3 } } },
   { DS_RDNS,  OM_S_OBJECT,  { { 0, rdn4 } } },
   OM_NULL_DESCRIPTOR
   };

   /*
    * Public Object ("Descriptor List") for Entry-Info-Selection
    * argument to ds_read( ).
    */
OM_descriptor  selection[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_ENTRY_INFORMATION_SELECTION),
   { DS_ALL_ATTRIBUTES,  OM_S_BOOLEAN,  { { OM_FALSE,  NULL } } },
   OM_OID_DESC(DS_ATTRIBUTES_SELECTED,  DS_A_TELEPHONE_NUMBER),
   { DS_INFORMATION_TYPE,OM_S_ENUMERATION,  { { DS_TYPES_AND_VALUES,NULL  } } },
   OM_NULL_DESCRIPTOR
   };

   /*
    * Variables to extract the telephone number(s).
    */

   OM_type            entry_list[ ]        = { DS_ENTRY,  0 };
   OM_type            attributes_list[ ]   = { DS_ATTRIBUTES,  0 };
   OM_type            telephone_list[ ]    = { DS_ATTRIBUTE_VALUES,  0 };
   OM_public_object   entry;
   OM_public_object   attributes;
   OM_public_object   telephones;
   OM_descriptor      * telephone;         /* current phone number */
```

```
   /*
    * Perform the Directory operations:
    * (1) Initialise the Directory Service and get an OM workspace.
    * (2) Bind a default directory session.
    * (3) Read the telephone number of "name".
    * (4) Terminate the directory session.
    */

   CHECK_DS_CALL( (OM_object) !(workspace=ds_initialize( )));

   CHECK_DS_CALL(ds_version(bdcp_package,  workspace));

   CHECK_DS_CALL(ds_bind(DS_DEFAULT_SESSION,  workspace, &session));

   CHECK_DS_CALL(ds_read(session,  DS_DEFAULT_CONTEXT,  name, selection,
                         &result, &invoke_id));

   /*
    * NOTE: check here for Attribute-Error (no-such-attribute)
    * in case the "name" doesn't have a telephone.
    * Then for all other cases call error_handler.
    */

   CHECK_DS_CALL(ds_unbind(session) );

   /*
    * Extract the telephone number(s) of "name" from the result.
    *
    * There are 4 stages:
    * (1) Get the Entry-Info from the Read-Result.
    * (2) Get the Attributes from the Entry-Info.
    * (3) Get the list of phone numbers.
    * (4) Scan the list and print each number.
    */


CHECK_OM_CALL(    om_get(result,

   OM_EXCLUDE_ALL_BUT_THESE_TYPES
   + OM_EXCLUDE_SUBOBJECTS,
   entry_list,  OM_FALSE,  0,  0,  &entry,  &total_number));

CHECK_OM_CALL(    om_get(entry->value.object.object,

   OM_EXCLUDE_ALL_BUT_THESE_TYPES
   + OM_EXCLUDE_SUBOBJECTS,
   attributes_list,  OM_FALSE,  0,  0,  &attributes,  &total_number));

CHECK_OM_CALL(    om_get(attributes->value.object.object,

   OM_EXCLUDE_ALL_BUT_THESE_TYPES
   + OM_EXCLUDE_SUBOBJECTS,
   telephone_list,  OM_FALSE,  0,  0,  &telephones,  &total_number));
```

```
/*
 *  We can now safely release all the private objects
 *  and the public objects we no longer need.
 */

CHECK_OM_CALL(om_delete(session) );
CHECK_OM_CALL(om_delete(result) );
CHECK_OM_CALL(om_delete(entry) );
CHECK_OM_CALL(om_delete(attributes) );

for (telephone = telephones;
     telephone->type == DS_ATTRIBUTE_VALUES;
     telephone++)
{
     if (telephone->type   != DS_ATTRIBUTE_VALUES
     || (telephone->syntax & OM_S_SYNTAX) != OM_S_PRINTABLE_STRING)
     {
          (void) fprintf(stderr, "malformed telephone number\n");
          exit(EXIT_FAILURE);
     }

     (void) printf("Telephone number: %s\n",
          telephone->value.string.elements);
}

CHECK_OM_CALL(om_delete(telephones) );

/*
 *  More application-specific processing can occur here ...
 */

/* ... and finally exit. */
exit(EXIT_SUCCESS);
}
```

A.3 Asynchronous Directory Example

```
/*
 *  Sample application that uses XDS in asynchronous mode.
 *
 *  The program adds a person as a member of a group.
 *
 *  This program uses the System V poll() function by way of
 *  example. Similar programs using BSD select() are possible.
 */

#include   <stdio.h>
#include   <fcntl.h>
#include   <poll.h>
#include   <signal.h>
#include   <assert.h>

#include   <xom.h>       /* Object Management header */
#include   <xds.h>       /* Directory Service header */
#include   <xdsbdcp.h>   /* Basic Directory Contents Package header */

#include   "example.h"   /* local macros and function prototypes */

/*
 *  Define necessary Object Identifier constants.
 */

OM_EXPORT(DS_A_MEMBER)
OM_EXPORT(DS_C_CONTEXT)
OM_EXPORT(DS_C_ENTRY_MOD)
OM_EXPORT(DS_C_ENTRY_MOD_LIST)

/*
 * Assume that we have built OM objects as follows:
 *
 * (1) "username" = distinguished name of user executing this program.
 * (2) "personname" = person to be added to directory.
 * (3) "groupname" = groupOfNames which "personname" will join.
 *
 * These would be built in the same way as the "name" argument
 * in the previous example.
 */

int asynchronous_function_example(

  OM_private_object  username,
  OM_private_object  personname,
  OM_private_object  groupname)

{
```

```
   DS_status         error;            /* return value from DS functions */
   OM_return_code    return_code;      /* return value from OM functions */
   OM_workspace      workspace;        /* workspace for directory operations */
   OM_private_object context;          /* context for directory operations */
   OM_private_object session;          /* session for directory operations */
   OM_sint           invoke_id;        /* Invoke-ID of the modify operation */
   OM_uint           completion_flag;  /* results of ds_receive_result()  */
   DS_status         operation_status; /*   "         */
   OM_private_object result;           /*   "         */
   OM_sint           result_id;        /*   "         */
   OM_value_position total_number      /* Number of Attribute Descriptors */

   static DS_feature bdcp_package[ ] = {
           {DS_BASIC_DIR_CONTENTS_PKG,  OM_TRUE},
           {{(OM_uint32)0,  (void *)0}, OM_FALSE}
   };

   /*
    *  Descriptors to set asynchronous mode and high priority in the context.
    */
OM_descriptor         context_options[ ] = {
   { DS_ASYNCHRONOUS,  OM_S_BOOLEAN,  { { TRUE,  NULL } } },
   { DS_PRIORITY,  OM_S_ENUMERATION,  { { DS_HIGH,  NULL } } },
   OM_NULL_DESCRIPTOR,
   };

   /*
    *  Descriptors used to build arguments for modify-entry operation.
    */
static OM_descriptor  modification[ ] = {
   OM_OID_DESC(OM_CLASS,           DS_C_ENTRY_MOD),
   OM_OID_DESC(DS_ATTRIBUTE_TYPE,  DS_A_MEMBER),
   { DS_ATTRIBUTE_VALUES,   OM_S_OBJECT,       { { 0, personname } } },
   { DS_MODIFICATION_TYPE,  OM_S_ENUMERATION,  { { DS_ADD_VALUES,  NULL } } },
   OM_NULL_DESCRIPTOR,
   };
OM_descriptor         changes[ ] = {
   OM_OID_DESC(OM_CLASS,  DS_C_ENTRY_MOD_LIST),
   { DS_CHANGES,  OM_S_OBJECT,  { { 0, (OM_object) modification } } },
   OM_NULL_DESCRIPTOR,
   };

   /*
    *  Arguments for om_get( ) to extract file descriptor from session.
    */

   OM_type         fd_list[ ]     = { DS_FILE_DESCRIPTOR,  0 };
   OM_descriptor   * fd_values;
```

```
/*
 * (1) Initialise the Directory Service and get an OM workspace,
 * (2) create a context with the required options,
 * (3) start a Directory session, and
 * (4) invoke the modification of the "groupname" entry.
 */

CHECK_DS_CALL( (OM_object) !(workspace=ds_initialize( )));
CHECK_DS_CALL(ds_version(bdcp_package,  workspace));

CHECK_OM_CALL(om_create(DS_C_CONTEXT,  OM_TRUE, workspace, &context);
CHECK_OM_CALL(om_put(context,  OM_REPLACE_ALL,  context_options,  NULL,
                           OM_ALL_VALUES,  OM_ALL_VALUES));

CHECK_DS_CALL(ds_bind(DS_DEFAULT_SESSION,  workspace, &session));
CHECK_DS_CALL(ds_modify_entry(session,  context, groupname, changes,
                  &results, &invoke_id));

/*
 * The application goes off and does some more stuff here ...
 */

/*
 * now go into poll( ) loop,
 *
 * (1) set file-descriptor of directory session into poll( ) list, and
 * (2) set to poll( ) on input events.
 */

CHECK_OM_CALL(om_get(session,  OM_EXCLUDE_ALL_BUT_THESE_TYPES,
                  fd_list,  OM_FALSE,  0,  0,  &fd_value,  &total_number))

pollfds[0].fd       = fd_value->value.integer;
pollfds[0].events   = POLLIN;

while (1)
{
     /* execute poll() */
     if (poll(pollfds,  NUM_FDS,  -1) < 0)
     {
          perror("poll failed");
          exit(1);
     }

     for (i = 0; i < NUM_FDS; i++)
     {
          switch (pollfds[i].revents)
          {
          case 0:
               continue;
               break;

          case POLLIN:
```

```
/*
 * There has been some activity on the directory session.
 * (1) ask for a result,
 * (2) check if an operation has completed,
 * (3) check whether it completed successfully,
 * (4) check if it was the operation we're interested in, and
 * (5) use the result (if it had been an interrogation).
 */

CHECK_DS_CALL(ds_receive_result(session,
                        &completion_flag,
                        &operation_status,
                        &result,
                        &result_id));

if (completion_flag != DS_COMPLETED_OPERATION)
                /*
                 * Whatever the file activity was, its not for us.
                 */
                break;

CHECK_DS_CALL(operation_status);

if (result_id != invoke_id)
{
                (void) fprintf(stderr, "Unexpected results!");
                exit(EXIT_FAILURE);
}

/*
 * If we get here, the Modify-Entry() operation has completed
 * successfully, and if it were an interrogation, the result
 * would be available for use now ...
 */

/*
 *  ... but there is no result from ds_modify_entry.
 */
assert(result == DS_NULL_RESULT);

break;

      /* Include other application events ... */

      default:
           perror("poll returned error event");
           }
      }
}
```

```
/*
 *  Terminate the directory session and clean up.
 */

CHECK_DS_CALL(ds_unbind(session) );

CHECK_OM_CALL(om_delete(context) );
CHECK_OM_CALL(om_delete(session) );
CHECK_OM_CALL(om_delete(result) );
CHECK_OM_CALL(om_delete(fd_value) );

/*
 * More application-specific  processing can occur here ...
 */

/*
 *  ... and exit.
 */
exit(EXIT_SUCCESS);

}
```

A.4 Error Handling Module

Each example program includes a local header, **<example.h>**, which defines macros used to check that functions in the directory interface and the object management interface all complete successfully. The contents of this header are reproduced below.

```
/*
 * Define some convenient exit codes.
 */

#define EXIT_FAILURE 1
#define EXIT_SUCCESS 0

/*
 * Declare an error handling function and an error checking macro for DS.
 */

void handle_ds_error(DS_status  error);

#define CHECK_DS_CALL(function_call)\
        error = (function_call);\
        if (error != DS_SUCCESS)\
        handle_ds_error(error);

/*
 * Declare an error handling function and an error checking macro for OM.
 */

void handle_om_error(OM_return_code  return_code);

#define CHECK_OM_CALL(function_call)\
        return_code = (function_call);\
        if (return_code != OM_SUCCESS)\
        handle_om_error(return_code);
```

As can be seen, the error checking macros make use of error handling functions in the event that an interface function did not complete successfully. Very simple versions of the error handling functions are reproduced below. They simply print an appropriate error message and terminate the program.

```
/*
 * The error handling code.
 *
 * NOTE: any errors arising in these functions are ignored.
 */
```

```
void handle_ds_error(DS_status  error)
{
   (void) fprintf(stderr, "DS error has occurred\n");

   (void) om_delete((OM_object) error);

   /*
    *  At this point, the error has been reported and storage cleaned up,
    *  so the handler could return to the main program now for it to take
    *  recovery action.
    */

   exit(EXIT_FAILURE);
}

void handle_om_error(OM_return_code  return_code)
{
   (void) fprintf(stderr, "OM error %d has occurred\n", return_code);

   /*
    *  At this point, the error has been reported and storage cleaned up,
    *  so the handler could return to the main program now for it to take
    *  recovery action.
    */

   exit(EXIT_FAILURE);
}
```

Appendix B

XDS Issue 2 and the IEEE DS Standard

This Appendix summarizes the substantive differences between the X/Open XDS Issue 2 CAE Specification and the IEEE DS Standard.

The IEEE has published its **Directory Services** Standard as IEEE 1224.2-1993 (language independent standard) and IEEE 1327.2-1993 (C binding). Development of this IEEE Standard was based on the X/Open **API to Directory Services (XDS)** CAE Specification (C190, November 1991) which is known as **XDS Issue 1**.

X/Open's intent in upgrading **XDS Issue 1** (C190) to **XDS Issue 2** (C317) was to align its XDS CAE specification with the corresponding IEEE DS Standard wherever possible. However, there were a few instances where full alignment was not achieved. These differences are described below.

B.1 DS_E_BAD_CLASS in Service Call Definitions

The Library Error **bad-class** is not listed under any of the interface operations of IEEE 1224.2, yet it may be returned if an argument of syntax **OM_object** is not of a supported class for an operation. In XDS, this error is returned by the following functions: *ds_add_entry*( ), *ds_compare*( ), *ds_list*( ), *ds_search*( ), *ds_modify_rdn*( ), and *ds_read*( ).

The omission of **bad-class** from the error lists of the P1224.2 operations would appear to be a defect in the IEEE 1224.2 Standard.

B.2 Correspondence of C Identifier Usage

Both XDS and IEEE 1224.2 describe how a reader can derive a C identifier from the language-independent name. The C identifiers in XDS and in IEEE 1327.2 have been shortened (from what was proposed originally) in order to keep them within a 32 character limit. In XDS, the language-independent names have been shortened to correspond. In IEEE 1224.2, they have not.

This difference does not affect the C binding, and therefore has no impact on conforming implementations or applications.

B.3 DL-Submit-Permission Member-of-Group

In IEEE 1224.2, the description of the **DL-Submit-Permission** class is confusing and inconsistent. The sentence

> *Note that exactly one of the four ...*

contradicts the attribute table. It is unclear whether the *Member-of-Group* attribute is also an OR-Name, and what its value number is.

This class definition is derived from the definition of the MHS DL submit permission attribute syntax in section A.3.1 of Annex A to CCITT Recommendation X.402 (1988), and the permissible attribute values should reflect that syntax definition.

In XDS, the inconsistency has been removed by:

- changing the value number of *Member-of-Group* from
  > 0 or more

  to
  > 0 or 1,
- changing
  > Note that exactly one of the four OR-Name attributes shall be present

  to
  > Note that exactly one of the four name attributes shall be present.

B.4 Numeric Values of Symbolic Constants

Use of a particular set of symbolic constants is mandatory in XDS but optional in IEEE 1327.2. Similarly, XDS gives precise definitions for macros and structures, while IEEE 1327.2 defines the functional effects of macros and lists the elements of structures without precluding the addition of further elements.

This difference means that there may be some P1224.2-compliant systems that are not XDS-compliant, but that all XDS-compliant systems will also be P1224.2 compliant. It does not affect well-behaved applications (that is, applications that use the symbolic constants rather than their numeric values.)

B.5 Use of errno

XDS specifies that, when a system error occurs, the value of the Problem attribute of the instance of class System-Error that is returned must be the same as errno. IEEE 1326.2 says that it must be the same as errno in POSIX-compliant systems, but does not constrain it otherwise.

Again, this difference means that there may be some IEEE 1224.2-compliant systems that are not XDS-compliant, but that all XDS-compliant systems will also be IEEE 1224.2 compliant. Moreover, the systems that comply with IEEE P1224.2 but not with XDS will not be POSIX-compliant, and are therefore not X/Open-compliant.

B.6 Internationalisation

Following X.520 (1988), XDS defines a number of Directory Attributes to have syntax `String(Teletex)`. This is not sufficiently general to support all character sets and codesets used internationally. This issue has been recognised by the standards bodies responsible for X.520. A new version of X.520 has been proposed and (in this respect at least) has been agreed from a technical point of view, but has not yet been formally adopted by the CCITT. This introduces a new syntax, `DirectoryString`, for these attributes. XDS includes support for this syntax, but IEEE 1224.2 does not.

The sections of XDS that describe support for `DirectoryString` syntax are as follows.

- In Section 7.2 on page 97:
  - — there is an explanation of the ASN.1 `DirectoryString` syntax, and of the meaning of `String(Directory)` in the XDS OM class descriptions
  - — general matching rules for `String(Directory)` attributes are described:
  - — in Table 7-2 on page 99, the OM Value Syntax is shown as `String(Directory)` in the entries for

    A-Business-Category
    A-Common-name
    A-Description
    A-Knowledge-Information
    A-Locality-Name
    A-Organization-Name
    A-Organizational-Unit-Name
    A-Physical-Delivery-Office-Name
    A-Post-Office-Box
    A-Postal-Code
    A-State-Or-Province-Name
    A-Street-Address
    A-Surname
    A-Title

    In IEEE 1224.2, they are shown as `String(Teletex)`.
- In Table 7-14 on page 115. the OM Value Syntax of Postal-Address is shown as `String(Directory)`. In IEEE 1224.2, it is shown as `String(Teletex)`.

Appendix C

XDS Issue 3 and the ISO DS Standard

The International Organization for Standardization (ISO) published its **Directory Services (DS) Standard** as four documents:

- ISO DIS 14392: Directory Services, Language-Independent Specification.
  Note that this document has been re-numbered from 14360.
- ISO DIS 14393: Directory Services, Test Methods for the Language-Independent Specification
- ISO DIS 14394: Directory Services, C Language Binding
- ISO DIS 14395: Directory Services, Test Methods for the C Language Binding.

These ISO draft international standards are based on the IEEE Standards (1224.2, 1326.2, 1327.2 and 1328.2), but the issues raised in the X/Open **XDS Issue 2** specification (API to Directory Services, Issue 2, C317) Appendix B were also taken into consideration. The outcome of this consideration is summarized below.

X/Open's intent in upgrading from the IEEE-aligned **XDS Issue 2** (C317) to **XDS Issue 3** (C608) is to align its XDS CAE specification with the corresponding ISO DS Standard.

This has been achieved except as identified below.

C.1 Outcome of Issues Raised in XDS Issue 2

The following items explain the outcome of all the issues raised in **XDS Issue 2** (C317) Appendix B.

IEEE Interpretation Requests

The issues described in sections B.1, B.2 and B.3 of Appendix B to the **XDS Issue 2** CAE Specification (C317) were also the subject of IEEE Interpretation Requests.

For C317 sections B.1 (DS_E_BAD_CLASS in Service Call Definitions) and B.3 (DL-Submit-Permission-Member-of-Group clarification), the draft international standards have been brought into line with what was defined in the **XDS Issue 2** specification.

For C317 section B.2, the issue relates to the fact that the C identifiers in the draft international standards have been shortened to ensure that they are unique within the first 32 characters, but the language-independent identifiers have not been shortened. In the X/Open XDS specification, both the C and the language-independent identifiers have been shortened. The change that has been made to the draft international standards is that the long language-independent identifiers have been kept but the description of the algorithm for converting language-independent identifiers to C identifiers has been changed. The language-independent identifiers in the draft international standards are therefore different from those in XDS. The C identifiers are the same, so there is no impact on implementations.

Remaining Issues

The issues described in C317 sections B.4, B.5 and B.6 have not been covered in the draft international standards. It was decided that the issue raised in B.5 (use of errno) should not be covered. The issues described in B.4 (Numeric Values of Symbolic Constants) and B.6 (Internationalisation) were not addressed because of the way that the ballot comments on the Directory Service API draft international standards were phrased.

Appendix D

Changes between XDS Issues 1/2/3

This Appendix summarizes the substantive differences between

- the **XDS Issue 1** CAE Specification C190 (November 1991) and the **XDS Issue 2** CAE Specification C317 (March 1994)
- the **XDS Issue 2** CAE Specification C317 (March 1994) and the **XDS Issue 3** CAE Specification C608.

D.1 Differences between XDS Issues 1 and 2

Add SAP classes for Certificates
: Definitions for Certificates, Cross-Certificates and Forward-Certification-Path were added to the Directory Class definitions Chapter.

Error *bad class* added
: Library error *bad class* was added to several functions to cover the eventuality that an argument of syntax **OM_object** may not be of a class supported for that function.

Correspondence of C-identifier names in Definitions & Headers
: The abbreviated C-identifier names that were adopted were harmonised between their definitions and their declarations in the header files.

Clarification for DL-Submit-Permission Member-of-Group
: The *Member-of-Group* definition for **DL-Submit-Permission** was corrected.

Support for Directory Strings
: A new version of X.520 has been proposed to define a number of Directory Attributes to have syntax *String***(Teletex)**. This change introduced a new syntax **Directory String** for these attributes. These changes appear mainly in the specification for Selected Attribute Types in the directory class definitions.

D.2 Difference between XDS Issues 2 and 3

Definition for AVA
: The definition given in the XDS Glossary for Attribute Value Assertion (AVA) is aligned with that given in Technical Corrigendum 2 to ISO / IEC 9594-2.

Glossary

This document makes use of many terms which are used to describe both the directory and object management; **Attribute** is an example. In these cases the two meanings of the term are indicated by annotation with (*Directory*) or (*Object*).

[*(D)* : reference adapted from the standards]

[*(O)* : reference adapted from the Object Management API Specification]

[*(P)* : reference adapted from POSIX.4 (real-time)]

[*(X)* : reference from the **X/Open Portability Guide**]

Abstract Class
An **OM Class** of **OM Object** of which instances are forbidden. An abstract class typically serves to document the similarities between instances of two or more **Concrete Classes**. *(O)*

Abstract Syntax Notation One
A notation which both enables complicated types to be defined and also enables values of these types to be specified. (See ISO Standard 8824, **Referenced Documents**.)

Access Point
The point at which an Abstract Service is obtained. (A connection between a DUA and a DSA.) *(D)*

Address
An unambiguous name, label or number which identifies the location of a particular entity or service. See also **Presentation Address**.

Alias, Alias Name
A **Name** for a (directory) **Object**, provided by the use of one or more **Alias Entries** in the DIT. *(D)*

Alias Entry
A directory entry, of **Object Class** 'alias', containing information used to provide an alternative name for an object. *(D)*

Argument
Information which is passed to a **Function** or **Operation** and which specifies the details of the processing to be performed.

ASN.1
See **Abstract Syntax Notation One**.

Asynchronous Completion
An asynchronous operation is complete when a corresponding synchronous operation would complete and any associated status fields have been updated. *(P)*

Asynchronous Operation
An operation that does not itself cause the process requesting the operation to be blocked from further use of the CPU. This implies that the process and the operation are running concurrently. *(P)*

Attribute (*Directory***)**
The information of a particular type concerning an object and appearing in an entry describing the object in the DIB. *(D)*

Attribute (*Object*)
See **OM Attribute**.

Attribute Syntax
A definition of the set of values which an **Attribute** may assume. It includes the data type, in ASN.1, and, usually, one or more *matching rules* by which values may be compared.

Attribute Type (*Directory*)
That component of an **Attribute** which indicates the class of information given by that attribute. It is an **Object Identifier**, and so completely unique. *(D)* i.

Attribute Type (*Object*)
Any of various categories into which the client dynamically groups values on the basis of their semantics. It is an integer, unique only within the **Package**. *(O)*

Attribute Value (*Directory*)
A particular instance of the class of information indicated by an **Attribute Type**. *(D)*

Attribute Value (*Object*)
An atomic information object. *(O)*

Attribute Value Assertion (AVA)
A proposition, which may be true, false or undefined, according to the specified matching rules for the type, concerning the presence in an entry of an attribute value (or a distinguished value) of a particular type.

Attribute Value Syntax
See **Syntax** (*Object*).

Basic Encoding Rules
A set of rules used to encode ASN.1 values as strings of octets.

BER
See **Basic Encoding Rules**.

Cache
See **Copy**.

CCITT
International Telephone and Telegraph Consultative Committee.
This organisation has now been renamed as the Telecommunications Standardization Section of the International Telecommunications Union (ITU-T).

Certificate
A synonym for **User Certificate**.

Chaining
A mode of interaction optionally used by a DSA which cannot perform an operation itself. The DSA chains by invoking an operation of another DSA and then relaying the outcome to the original requestor. *(D)*

Class (*Directory*)
See **Object Class**.

Class (*Object*)
See **OM Class**.

Continuation Reference
A Continuation Reference describes how the performance of all or part of an **Operation** can be continued at a different DSA or DSAs. See also **Referral**. *(D)*

Copy
Either a copy of an entry stored in other DSA(s) through bilateral agreement, or a locally and dynamically stored copy of an entry resulting from a request (a cache copy). *(D)*

Concrete Class
An OM class of which instances are permitted. *(O)*

Descriptor
A defined data structure which is used to represent an OM **Attribute Type** and a single value.

Descriptor List
An ordered sequence of **Descriptors** which is used to represent several OM **Attribute Types** and **Attribute Values**.

Directory
A collection of open systems which cooperate to hold a logical database of information about a set of objects in the real world. *(D)*

Directory Information Base (DIB)
The complete set of information to which the directory provides access and which includes all the pieces of information which can be read or manipulated using the operations of the directory. It is made up of **Entries**. *(D)* i.

Directory Information Tree (DIT)
The DIB considered as a tree, whose vertices (other than the root) are the directory **Entries**. *(D)*

Directory System Agent (DSA)
An OSI-application-process which is part of the directory. *(D)*

Directory User Agent (DUA)
An OSI-application-process which represents a user accessing the directory. (Note that this may be composed of an arbitrary number of system processes and application **Processes**, including the one or more which are the user). *(D)*

Distinguished Encoding
Restrictions to the **Basic Encoding Rules** designed to ensure a unique encoding of each ASN.1 value, defined in Clause 8.7 of ISO Standard 9594-8 (see **Referenced Documents**).

Distinguished Name
One of the names of an object, formed from the sequence of **RDN**s of its object **Entry** and each of its superior entries. *(D)*

Distinguished Value
An **Attribute Value** in an **Entry** which has been designated to appear in the RDN of the entry. *(D)*

Entry
The part of the DIB containing information relating to a single (directory) **Object**. Each entry is made up of (directory) **Attributes**. *(D)*

Filter
An assertion about the presence or value of certain attributes of an entry in order to limit the scope of a search. *(D)*

Function
A programming language construct, modelled after the mathematical concept. A function encapsulates some behaviour. It is given some arguments as input, performs some processing, and returns some results. Also known as procedures, subprograms or subroutines. See **Operation**.

Immediate Subordinate
In the DIT, an **Entry** is an immediate subordinate of another if its **Distinguished Name** is formed by appending its RDN to the distinguished name of the other entry.

Immediate Superior
In the DIT, an **Entry** is the immediate superior of another if its **Distinguished Name**, followed by the RDN of the other, forms the distinguished name of the other entry.

Implementation-Defined
The feature is not consistent across all implementations, and each implementation will provide documentation of its behaviour. *(X)*

Invoke-ID
An integer used to distinguish one (directory) **Operation** from all other outstanding ones.

ITU-T
Telecommunications Standardization Section of the International Telecommunications Union (ITU-T).
Re-named from the original CCITT.

Knowledge Reference
Knowledge which associates, either directly or indirectly, a DIT entry with the DSA in which it is located. *(D)*

Leaf Entry
A directory **Entry** which has no **Subordinates**. It can be an **Alias Entry** or an **Object Entry**.

Locally Administered
The configuration is not consistent across all systems, and the administrator of each system will provide documentation of its behaviour.

May
With respect to implementations, the feature is optional. Applications should not rely on the existence of the feature. *(X)*

Name
A construct that singles out a particular (directory) object from all other objects. A name must be unambiguous (that is, denote just one object), however it need not be unique (that is, be the only name which unambiguously denotes the object).

Non-specific Subordinate Reference
A **Knowledge Reference** that holds information about the DSA that holds one or more unspecified subordinate entries. *(D)*

Object (*Directory***)**
Anything in some 'world', generally the world of telecommunications and information processing or some part thereof, which is identifiable (can be named), and which it is of interest to hold information on in the DIB. *(D)*

Object (*Object***)**
An object is a composite information object comprising zero or more **OM Attributes** of different types.

Object Class
An identified family of **Objects** (or conceivable objects) which share certain characteristics. (See **Class**). *(D)*

Object Entry
See **Entry**.

Object Identifier
A value (distinguishable from all other such values) which is associated with an information object. *(X.208)*

OM Attribute
An OM attribute comprises one or more values of a particular type (and therefore syntax). *(O)*

OM Class
A static grouping of OM objects, within a specification, based on both their semantics and their form. *(O)*

Operation
Processing performed within the directory to provide a service, such as a read operation. It is given some arguments as input, performs some processing, and returns some results. An application process invokes an operation by calling an interface **Function**.

Outstanding Operation
An **Operation**, invoked asynchronously (that is, with **Asynchronous**=**true** in the context), which has not yet been the subject of a call to **Abandon()** or **Receive-Results()**.

Package
A specified group of related **OM Classes**, denoted by an **Object Identifier**.

Presentation Address
An unambiguous name which is used to identify a set of presentation-service-access-points. Loosely, it is the network address of an OSI service.

Private Object
An **OM object** created in a **Workspace** using the object management functions. The term is simply used for contrast with a **Public Object**.

Process
An address space, a single thread of control that executes within that address space, and its required system resources. As opposed to a 'system process', or the OSI usage of the term 'application process'. On a system that implements **Threads**, a process is redefined to consist of an address space with one or more threads executing within that address space and their required system resources. *(P)*

Public Object
A descriptor list which contains all the **OM Attributes** of an **OM Object**.

Purported Name
A construct which is syntactically a **Name** but which has not (yet) been shown to be a valid name. *(D)*

Referral
An outcome which can be returned by a DSA which cannot perform an operation itself, and which identifies one or more other DSAs more able to perform the operation. *(D)*

Relative Distinguished Name (RDN)
A set of **Attribute Value Assertions** (AVAs), each of which is true, concerning the distinguished values of a particular **Entry**. *(D)*

Replication
The process by which **Copies** of **Entries** are created and maintained.

Result
Information which is returned from a **Function** or **Operation** and which constitutes the outcome of the processing which was performed.

Schema
The Directory Schema is the set of rules and constraints concerning DIT structure, object class definitions, attribute types and syntaxes which characterise the DIB. *(D)*

Service Controls
A group of parameters applied to all directory operations, which direct or constrain the provision of the service. *(D)*

Session
A sequence of directory operations requested by a particular user of a particular DUA, using the same Session OM object.

Should
With respect to implementations, the feature is recommended, but it is not a mandatory requirement. Applications should not rely on the existence of the feature.

With respect to applications, the word is used to give guidelines for recommended programming practice. These guidelines should be followed if maximum portability is desired. *(X)*

Signed
Information is digitally signed by appending to it an enciphered summary of the information. This is used to ensure the integrity of the data, the authenticity of the originator, and the unambiguous relationship between the originator and the data. *(D)*

Subordinate
In the DIT, an **Entry** is subordinate to another if its **Distinguished Name** includes that of the other as a prefix.

Superior
In the DIT, an **Entry** is superior to another if its **Distinguished Name** is included as a prefix of the distinguished name of the other. Each entry has exactly one immediate superior.

Syntax (*Directory***)**
See **Attribute Syntax**.

Syntax (*Object***)**
An OM syntax is any of various categories into which the Object Management Specification statically groups values on the basis of their form. These categories are additional to the OM type of the value. *(O)*

Thread
A single sequential flow of control within a **Process**. *(P)*

Undefined
A feature is undefined if this document imposes no portability requirements on applications for erroneous program construct or erroneous data. Implementations *may* specify the result of using the feature, but such specifications are not guaranteed to be consistent across all implementations. That is, it is a programming error to use the feature, unless the particular implementation specifies the result. Note that an undefined result is completely unpredictable and may include abnormal program termination. *(X)*

Unspecified
A feature is unspecified if this document imposes no portability requirements on applications for correct program construct or erroneous data. Implementations *may* specify the result of using the feature, but such specifications are not guaranteed to be consistent across all implementations. That is, it is always permissible to use the feature, but the result is not known unless specified by the particular implementation. *(X)*

User
The end user of the directory; the entity or person which accesses the directory. Refers here to the application program which is calling the interface. *(D)*

User Certificate
The public keys of a user, together with some other information, rendered unforgeable by encipherment with the secret key of the certification authority which issued it. *(D)*

Value
See **Attribute Value**.

Will
The feature is required to be implemented and applications can rely on its existence. *(X)*

Workspace
A space in which **OM objects** of certain **OM classes** can be created, together with an implementation of the object management functions which supports those OM classes.

Index