

IEEE Standard for Information Technology—Directory Services—Application Program Interface (API) [Language Independent]

Sponsor
**Portable Applications Standards Committee
of the
IEEE Computer Society**

Approved March 18, 1993
IEEE Standards Board

Approved September 16, 1993
American National Standards Institute

Abstract: A application program interface (API) to OSI directory services is defined in terms that are independent of any particular programming language. The interface, which supports applications portability at the source-code level, is intended to be used by both applications developers and directory service implementors.

Keywords: API, application portability, ASN.1, directory, open systems, portable application, POSIX

Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1994 by the
Institute of Electrical and Electronics Engineers, Inc.

All rights reserved. Published 1994

Printed in the United States of America

ISBN 1-55937-303-2

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

IEEE standards documents may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers, Inc. does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Contents

	PAGE
Introduction	vii
Section 1: General	1
1.1 Scope	1
1.2 Normative References	2
1.3 Conformance	4
1.4 Test Methods	7
Section 2: Terminology and General Requirements	9
2.1 Conventions	9
2.2 Definitions	11
Section 3: Description	23
3.1 General	23
3.2 Initialization and Shutdown	24
3.3 Session	25
3.4 Context	26
3.5 Operation Parameters	26
3.6 Operation Results	29
3.7 Synchronous and Asynchronous Directory Operations	31
3.8 Automatic Connection Management	32
3.9 Automatic Continuation and Referral Handling	33
Section 4: Interface Operations	35
4.1 Datatype Definitions	35
4.2 Abandon	36
4.3 Add Entry	37
4.4 Bind	39
4.5 Compare	40
4.6 Initialize	42
4.7 List	43
4.8 Modify Entry	45
4.9 Modify RDN	47
4.10 Read	48
4.11 Receive Result	50
4.12 Remove Entry	53
4.13 Search	54
4.14 Shutdown	56
4.15 Unbind	57
4.16 Version	58

Section 5: Interface Class Definitions	61
5.1 Overview	61
5.2 Class Hierarchy	61
5.3 Access Point	62
5.4 Address	63
5.5 Attribute	64
5.6 Attribute List	65
5.7 AVA	65
5.8 Common Results	66
5.9 Compare Result	66
5.10 Context	67
5.11 Continuation Reference	72
5.12 DS DN	73
5.13 DS RDN	73
5.14 Entry Information	74
5.15 Entry Information Selection	75
5.16 Entry Modification	77
5.17 Entry Modification List	78
5.18 Extension	78
5.19 Filter	79
5.20 Filter Item	81
5.21 List Info	83
5.22 List Info Item	84
5.23 List Result	85
5.24 Name	86
5.25 Operation Progress	86
5.26 Partial Outcome Qualifier	88
5.27 Presentation Address	89
5.28 Read Result	90
5.29 Relative Name	90
5.30 Search Info	91
5.31 Search Result	92
5.32 Session	93
Section 6: Error Class Definitions	95
6.1 Introduction	95
6.2 OM Class Hierarchy	96
6.3 Error	97
6.4 Abandon Failed	98
6.5 Attribute Error	99
6.6 Attribute Problem	100
6.7 Communications Error	101
6.8 Library Error	102
6.9 Name Error	103
6.10 Referral	104
6.11 Security Error	105
6.12 Service Error	106
6.13 System Error	107
6.14 Update Error	108

Section 7: The X.500 Directory Service	109
7.1 Introduction	109
7.2 Package Object Identifiers	110
7.3 Selected Attribute Types	110
7.4 Selected Object Classes	117
7.5 OM Class Hierarchy	118
7.6 Algorithm Identifier	121
7.7 Certificate	121
7.8 Certificate List	123
7.9 Certificate Pair	123
7.10 Certificate Sublist	124
7.11 Certificates	125
7.12 Cross-Certificates	126
7.13 DL Submit Permission	126
7.14 Facsimile Telephone Number	127
7.15 Forward-Certification-Path	128
7.16 Postal Address	129
7.17 Search Criterion	130
7.18 Search Guide	131
7.19 Signature	132
7.20 Teletex Terminal Identifier	132
7.21 Telex Number	133
Section 8: Definitions of Constants	135
8.1 Introduction	135
8.2 Directory Service Package	135
8.3 Basic Directory Contents Package	135
8.4 Strong Authentication Package	136
8.5 MHS Directory User Package	136
Annex A (informative) Concepts and Models	141
A.1 Overview	141
A.2 Directory Model	141
A.3 OM Concepts	143
A.4 Security	147
Annex B (informative) Bibliography	149
Alphabetic Topical Index	152

FIGURES

Figure A-1 – Example Directory Distinguished Name	142
Figure A-2 – Overview of Directory Components	143

TABLES

Table 3-1 – Interface Operations Summary	24
Table 5-1 – OM Attributes of Access-Point	63
Table 5-2 – OM Attributes of Attribute	64
Table 5-3 – OM Attributes of Attribute-List	65
Table 5-4 – OM Attributes of Common-Results	66
Table 5-5 – OM Attributes of Compare-Result	67
Table 5-6 – OM Attributes of Context	68
Table 5-7 – OM Attributes of Continuation-Reference	72
Table 5-8 – OM Attributes of DS-DN	73
Table 5-9 – OM Attributes of DS-RDN	74
Table 5-10 – OM Attributes of Entry-Information	75
Table 5-11 – OM Attributes of Entry-Information-Selection	76
Table 5-12 – OM Attributes of Entry-Modification	77
Table 5-13 – OM Attributes of Entry-Modification-List	78
Table 5-14 – OM Attributes of Extension	79
Table 5-15 – OM Attributes of Filter	80
Table 5-16 – OM Attributes of Filter-Item	81
Table 5-17 – OM Attributes of List-Info	83
Table 5-18 – OM Attributes of List-Info-Item	84
Table 5-19 – OM Attributes of List-Result	85
Table 5-20 – OM Attributes of Operation-Progress	87
Table 5-21 – OM Attributes of Partial-Outcome-Qualifier	88
Table 5-22 – OM Attributes of Presentation-Address	89
Table 5-23 – OM Attributes of Read-Result	90
Table 5-24 – OM Attributes of Search-Info	91
Table 5-25 – OM Attributes of Search-Result	92
Table 5-26 – OM Attributes of Session	93
Table 6-1 – OM Attributes of Error	97
Table 6-2 – OM Attributes of Attribute-Error	99
Table 6-3 – OM Attributes of Attribute-Problem	100
Table 6-4 – OM Attributes of Name-Error	103
Table 7-1 – Object Identifiers for Selected Attribute Types	116
Table 7-1 – Object Identifiers for Selected Attribute Types (<i>concluded</i>)	117
Table 7-2 – Representation of Values for Selected Attribute Types	119
Table 7-2 – Representation of Values for Selected Attribute Types (<i>concluded</i>)	120
Table 7-3 – Object Identifiers for Selected Object Classes	120

Table 7-4 – OM Attributes of Algorithm-Identifier	121
Table 7-5 – OM Attributes of Certificate	122
Table 7-6 – OM Attributes of Certificate-List	123
Table 7-7 – OM Attributes of Certificate-Pair	124
Table 7-8 – OM Attributes of Certificate-Sublist	124
Table 7-9 – OM Attributes of Certificates	125
Table 7-10 – OM Attributes of Cross-Certificates	126
Table 7-11 – OM Attributes of DL-Submit-Permission	127
Table 7-12 – OM Attributes of Facsimile-Telephone-Number	128
Table 7-13 – OM Attributes of Forward-Certificate-Path	129
Table 7-14 – OM Attributes of Postal-Address	129
Table 7-15 – OM Attributes of Search-Criterion	130
Table 7-16 – OM Attributes of Search-Guide	131
Table 7-17 – OM Attributes of Signature	132
Table 7-18 – OM Attributes of Teletex-Terminal-Identifier	133
Table 7-19 – OM Attributes of Telex-Number	134
Table 8-1 – DSP OM Class Object Identifier Suffixes	136
Table 8-2 – DSP OM Attribute Type Identifiers	137
Table 8-3 – BDCP OM Class Object Identifier Suffixes	137
Table 8-4 – BDCP OM Attribute Type Identifiers	138
Table 8-5 – SAP OM Class Object Identifier Suffixes	138
Table 8-6 – SAP OM Attribute Type Identifiers	138
Table 8-7 – MDUP OM Class Object Identifier Suffixes	139
Table 8-8 – MDUP OM Attribute Type Identifiers	139

Introduction

(This introduction is not a normative part of IEEE Std 1224.2-1993, IEEE Standard for Information Technology—Directory Services—Application Program Interface (API) [Language Independent], but is included for information only.)

Purpose

The purpose of this standard is to define an application programming interface (API) to directory services, including, but not limited to, the services described in ISO/IEC 9594: 1990 and the CCITT X.500 Series recommendations (References {5–11}) in terms that are independent of any particular programming language.

Related Standards

This standard is intended to provide the basis for the definition of programming language bindings to which implementations and applications can conform. Such a language binding to this standard, for the C programming language, is contained in IEEE Std 1327.2-1993 {B15}.

The API that this standard defines uses the mechanism for OSI abstract data manipulation (OM) defined in IEEE Std 1224-1993 {14}.

This standard, and the language bindings derived from it, are intended to be used in conjunction with related APIs, such as the API to X.400-based messaging services defined in IEEE Std 1224.1-1993 {15}.

IEEE Std 1326.2-1993 {16} contains a set of requirements to be satisfied by test methods for measuring conformance to this standard. They are stated in terms that are independent of any particular programming language, and they apply to test methods for measuring conformance to all standards defining programming language bindings of this standard.

Overview

This standard defines a mandatory generic interface to directory services and an optional extension of that interface for use in conjunction with the service described in ISO/IEC 9594 : 1990 and the CCITT X.500 Series recommendations (References {5–11}).

Acknowledgments

The base document for this standard is the *X/Open Directory Services API Specification (XDS)* {B19}, copyright © 1991 by X.400 API Association, United States of America and X/Open Company Limited, United Kingdom. The working group is grateful to both X/Open and the X.400 API Association for permission to use their materials.

Related Standards Activities

The following areas are under active consideration at this time, or are expected to become active in the near future, concerning standards for APIs that are related to the API defined in this standard. Similar efforts can be anticipated in the future.¹

- 1) X.400-based message handling
- 2) FTAM API
- 3) Other OSI application layer APIs
- 4) Network interface facilities
- 5) System administration

¹A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903, USA; Telephone: (+1) 202 371-0101; Fax: (+1) 202 728-9614.

Extensions are approved as “amendments” or “revisions” to this document, following IEEE Standards procedures.

Approved supplements are published separately and distributed with orders from the IEEE for this document until the full document is reprinted and such supplements are incorporated in their proper positions.

If you have any questions about the completeness of your version, you may contact the IEEE Standards Department [(+1) 908 562-3800] to determine what supplements have been published.

If you have an interest in participating in the working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronic Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA, and ask to have this forwarded to the chairperson of the appropriate PASC working group.

This standard was prepared by the Namespace and Directory Services Working Group (P1224.2, formerly P1003.17), sponsored by the Portable Applications Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the P1224.2 working group was as follows:

Portable Applications Standards Committee (PASC)

Chair:	Jim Isaak
Vice Chairs:	Hal Jespersen Lorraine Kevra Pete Meier Andrew Twigger
Treasurer:	Peter Smith
Secretary:	Shane McCarron

P1224.2 Working Group Officials

Chair:	Mark Hazzard
Vice Chair:	Donna Fisher
Editor:	Chris Harding
Secretary:	Donna Fisher

P1224.2 Working Group

Greg Bussiere
Scott Guthery

Edmun Liu
Ted Roberts

Rob Spade
Alec Tashjian

The following persons were members of the P1224.2 balloting group that approved this standard for submission to the IEEE Standards Board:

Stephen R. Walli (*EurOpen Organizational Representative*)

Ralph Barker (*Uniform Organizational Representative*)

Reed M. Adams	Michel Gien	Richard Seibel
Charles R. Arnold	Steve Griesmer	Ronald K. Sellars
Timothy Baker	Charles Hammons	Rob Spade
Ralph Barker	Michael Hannah	Sandra Swearingen
Andy Bihain	C. J. Harding	James G. Tanner
Mack W. Bishop	Mark Hazzard	Alec Tashjian
Robert Bismuth	Barry Hedquist	Donn S. Terry
Kevin Brady	Khiem Ho	Alan E. Turner
Dawn Burnett	Petr JaneDCL135ek	Andrew T. Twigger
Gregory Bussiere	Hal Jespersen	Mark-Rene Uchida
George Carson	Robert Knighten	Andrew Walker
Andy Cheese	Kevin Lewis	Paul Wanish
Fred D. Cowner	Roger Martin	Peter Wishart
Ana Maria De Alvare	Mark Modig	Oren Yuen
Stephen L. Diamond	Diana Norwood	John Zenor
Philip H. Enslow	Edward Owens	George R. Zerdian
Donna K. Fisher	Scott E. Preece	Jason Zions
Donald Folland	Paul Rabin	John Zolnowsky
	Seth Rosenthal	

The IEEE Standards Board approved this standard on March 18, 1993 with the following membership:

Wallace S. Read, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril	Ben C. Johnson	Don T. Michael*
Clyde R. Camp	Walter J. Karplus	Marco W. Migliaro
Donald C. Fleckenstein	Lorraine C. Kevra	L. John Rankine
Jay Forster*	E. G. "Al" Kiener	Arthur K. Reilly
David F. Franklin	Ivor N. Knight	Ronald H. Reimer
Ramiro Garcia	Joseph Koepfinger*	Gary S. Robinson
Donald N. Heirman	D. N. "Jim" Logothetis	Leonard L. Tripp
Jim Isaak		Donald W. Zipse

* Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal	Richard B. Engelman	David E. Soffrin
James Beall		Stanley I. Warshaw
	Mary Lynne Nielsen	
	<i>IEEE Standards Project Editor</i>	

IEEE Standard for Information Technology—Directory Services—Application Program Interface (API) [Language Independent]

Section 1: General

1.1 Scope

A directory is a distributed collection of information, which programs can access in order to make queries or updates. This standard defines an application programming interface (API) to directory services. This API is known as the directory services API (DS API). It is referred to as *the interface* throughout this standard.

This standard defines a standard directory service interface to support application portability at the source-code level. It is intended to be used by both application developers and directory service implementors.

The interface defined in this standard is intended to be used to provide access to a range of directory services that are instances of a common abstract model. That model is defined in the 1988 CCITT X.500 Series recommendations and ISO/IEC 9594: 1990. The CCITT Recommendations and ISO/IEC 9594: 1990 {5–11}¹⁾ were developed in close collaboration and are technically aligned. Hereinafter, they are referred to as *the X.500 directory standards*. A summary description of the model is contained in A.2.

This standard prescribes how the interface that it defines is to be used to access the particular directory service defined in ISO/IEC 9594: 1990 {5–11} and indicates

1) The numbers in curly brackets correspond to those of the references in 1.2. When preceded by a “B,” the numbers correspond to those of the bibliography in Annex B.

how it may be used to access other particular directory services that conform to the same abstract model.

This standard defines an API that application programs can use to access the functionality of the underlying directory service. It does not define or imply any profile of that service.

Nothing in this standard requires that the implementation of the interface or the directory itself actually make use of the Directory Access Protocol (DAP), the Directory System Protocol (DSP), or other parts of the model, just so 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 directory system agents (DSAs) are accessible.

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

It is intended that an application program should be able to use the interface defined in this standard to access a single directory service or to access several directory services at the same time.

This standard uses concepts of OSI abstract data manipulation (OM) that are defined in IEEE Std 1224-1993 {14}. They are summarized in A.3.

The directory service interface is defined in this standard in a programming language independent manner. It contains:

- Definitions of the interface operations
- Definitions of the information structures used in the interface that are associated with the abstract model of directory services
- Definitions of the information structures used in the interface that are associated with the particular instance of the model that is defined in ISO/IEC 9594: 1990 {5}-{11}.

This standard describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

This standard is defined exclusively at the source-code level. The objective is that a conforming application source program can be translated to execute in conjunction with one or more conforming implementations.

1.2 Normative References

The following standards contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to

agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

- {1} ISO/IEC 8824: 1990 (CCITT Recommendation X.208: 1988),²⁾ *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1).*
- {2} ISO/IEC 8825: 1990 (CCITT Recommendation X.209: 1988), *Information technology—Open Systems Interconnection—Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).*
- {3} ISO/IEC 9072-1: 1989 (CCITT Recommendation X.219: 1988), *Information processing systems—Text communication—Remote Operations—Part 1: Model, notation and service definition.*
- {4} ISO/IEC 9072-2: 1989 (CCITT Recommendation X.229: 1988), *Information processing systems—Text communication—Remote Operations—Part 2: Protocol specification.*
- {5} ISO/IEC 9594-1: 1990 (CCITT Recommendation X.500: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 1: Overview of concepts, models and services.*
- {6} ISO/IEC 9594-2: 1990 (CCITT Recommendation X.501: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 2: Models.*
- {7} ISO/IEC 9594-3: 1990 (CCITT Recommendation X.511: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 3: Abstract service definition.*
- {8} ISO/IEC 9594-4: 1990 (CCITT Recommendation X.518: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 4: Procedures for distributed operation.*
- {9} ISO/IEC 9594-6: 1990 (CCITT Recommendation X.520: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 6: Selected attribute types.*
- {10} ISO/IEC 9594-7: 1990 (CCITT Recommendation X.521: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 7: Selected object classes.*
- {11} ISO/IEC 9594-8: 1990 (CCITT Recommendation X.509: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 8: Authentication framework.*

²⁾ ISO/IEC documents can be obtained from the ISO office, 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse. CCITT documents can be obtained from the CCITT General Secretariat, International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

- 1 {12} CCITT Recommendation F.200: 1984, *Teletex Service*. CCITT Red Book,
2 Fascicle II.5.
- 3 {13} CCITT Series T Recommendations: Terminal Equipment and Protocols for
4 Telematic Services, Fascicle VII.3, 1984.
- 5 {14} IEEE Std 1224-1993,³⁾ *IEEE Standard for Information Technology—Open*
6 *Systems Interconnection (OSI) Abstract Data Manipulation—Application*
7 *Program Interface (API) [Language Independent]*.
- 8 {15} IEEE Std 1224.1-1993, *IEEE Standard for Information Technology—*
9 *X.400-Based Electronic Messaging—Application Program Interface (API)*
10 *[Language Independent]*.
- 11 {16} IEEE Std 1326.2-1993, *IEEE Standard for Information Technology—Test*
12 *Methods for Measuring Conformance to Directory Services—Application Pro-*
13 *gram Interface (API) [Language Independent]*.

14 1.3 Conformance

15 1.3.1 Implementation Conformance

16 1.3.1.1 Conformance Requirements

17 A conforming implementation for a programming language binding specification
18 for this standard shall meet all of the following criteria:

- 19 (1) The implementation shall support all required behavior defined in this
20 standard.
- 21 (2) The implementation shall support all required interfaces defined in the
22 programming language binding specification. Those interfaces shall sup-
23 port the behavior described in this standard and in the programming lan-
24 guage specification.
- 25 (3) The implementation may provide additional functions or facilities not
26 required by this standard or by the programming language binding
27 specification. Each such nonstandard extension shall be identified as
28 such in the system documentation. Nonstandard extensions, when used,
29 may change the behavior of functions or facilities defined by this standard
30 or by the programming language binding specification. The conformance
31 document shall define an environment in which an application can be run
32 with the behavior specified by this standard and the programming

33
34 3) IEEE documents are available from the Institute of Electrical and Electronics Engineers, Service Center, 445
35 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

1 language binding specification. In no case shall such an environment
2 require modification of a Strictly Conforming Application.

3 **1.3.1.2 Conformance Documentation**

4 A conformance document with the following information shall be available for an
5 implementation claiming conformance to a programming language binding
6 specification for this standard. The conformance document shall be in two parts.
7 The first part shall have the same structure as this standard, with the information
8 presented in the appropriately numbered sections, clauses, and subclauses. The
9 second part shall have the same structure as the programming language binding
10 specification, with the information presented in the appropriately numbered sec-
11 tions, clauses, and subclauses. The conformance document shall not contain infor-
12 mation about extended features or capabilities outside the scope of this standard
13 and the programming language binding specification.

14 The conformance document shall identify the programming language binding
15 specification to which the implementation conforms.

16 The conformance document shall contain a statement that indicates the full
17 names, numbers, and dates of the language-independent and programming lan-
18 guage binding specification standards that apply.

19 The conformance document shall state which of the optional features defined in
20 this standard and in the programming language binding specification are sup-
21 ported by the implementation.

22 The conformance document shall describe the behavior of the implementation for
23 all implementation-defined features defined in this standard and in the program-
24 ming language binding specification. This requirement shall be met by listing
25 these features and by providing either a specific reference to the system documen-
26 tation or full syntax and semantics of these features. The conformance document
27 may specify the behavior of the implementation for those features where this stan-
28 dard or the programming language binding specification states that implementa-
29 tions may vary or where features are identified as undefined or unspecified.

30 No specifications other than those specified by this standard and the programming
31 language binding specification shall be present in the conformance document.

32 The phrases “shall document” or “shall be documented” in this standard or in a
33 programming language binding specification for this standard mean that docu-
34 mentation of the feature shall appear in the conformance document, as described
35 previously, unless the system documentation is explicitly mentioned.

36 The system documentation should also contain the information found in the con-
37 formance document.

38 **1.3.1.3 Conforming Implementation Options**

39 All the interface operations are mandatory; none are optional.

Some OM attributes are optional; these are marked as such in the OM class definitions. They are

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

Some items of behavior of the interface are implementation defined. These are

- The maximum number of outstanding asynchronous operations
- Whether an asynchronous interface operation returns control before the directory operation is submitted to the Directory
- The text and language of error messages
- The provision or not of automatic connection management
- Whether `ds_abandon` issues an **Abandon** directory operation or not
- The OM classes permitted as values of the *Name* parameter to interface with operations

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

This standard defines four packages. The Directory Service package defined in Section 5, which also includes the errors defined in Section 6, is mandatory and shall be provided in each conforming implementation. The following three packages are optional:

- The Basic Directory Contents package defined in Section 7
- The Strong Authentication package defined in Section 7
- The MHS Directory User package defined in Section 7

Use of the optional packages is negotiated using the `ds_version` operation.

Section 8 contains a set of definitions of values for the OM class object identifiers and OM attribute type identifiers used in this standard. Use of these definitions by an implementation is optional. If an implementation is to interwork with other implementations of this standard or of related standards (such as the API to X.400-based electronic messaging defined in IEEE Std 1224.1-1993 [15]), then it must use the definitions given in Section 8 of this standard.

NOTE: This standard does not mandate that it be possible to encode any OM classes using `om_encode` and `om_decode`.

1.3.2 Application Conformance

All applications claiming conformance to a programming language binding specification for this standard shall fall within one of the categories defined in the following subclauses.

1.3.2.1 Strictly Conforming Application

A Strictly Conforming Application is an application that requires only the facilities described in this standard, in the programming language binding specification, and in the applicable language standards. Such an application shall accept any behavior described in this standard or in the programming language binding specification as unspecified or implementation defined and, for symbolic constants, shall accept any value in the ranges permitted by this standard and the programming language binding specification.

1.3.2.2 Conforming Application

There is only one type of Conforming Application.

1.3.2.2.1 ISO/IEC Conforming Application

An ISO/IEC Conforming Application of a programming language binding specification for this standard is an application that uses only the facilities described in this standard, in the programming language binding specification, in the applicable language standards, and in other language binding standards approved by ISO or IEC. Such an application shall include a statement of conformance that documents all options and limit dependencies, and all other ISO or IEC standards used.

1.3.2.3 Conforming Application Using Extensions

A Conforming Application Using Extensions of a programming language binding specification for this standard is an application that differs from a Conforming Application only in that it uses nonstandard facilities that are consistent with this standard and with the programming language binding specification. Such an application shall fully document its requirements for these extended facilities, in addition to the documentation required of a Conforming Application.

1.4 Test Methods

Any measurement of conformance to a programming language binding specification for this standard shall be performed using test methods that conform to IEEE Std 1326.2-1993 {16} and to any additional requirements that may be imposed by the programming language binding specification.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 General and Typographical Conventions

Many technical terms, such as “object” and “attribute,” are used by both OSI abstract data manipulation and directory services descriptions. In the first case, they refer to constructs of the OSI abstract data manipulation interface, while in the second case they refer to constructs of the directory service to which the interface provides access. The meanings ascribed to these terms in these cases are often similar but different. Throughout this standard, care is taken to distinguish between these meanings, for example by qualifying the term with “OM” or “directory” as appropriate (as with OM classes and directory classes and with OM attributes and directory attributes). The unqualified term “attribute” denotes the directory construct, while the phrase “OM attribute” denotes the OSI abstract data manipulation construct. The phrase “Object Class” denotes the directory construct, while the phrase “OM class” denotes the OSI abstract data manipulation construct.

The term “operation” is used in two different senses, one by the language-independent specification conventions and one by the directory service descriptions. When used in the sense of the directory, it is always qualified by the term “directory,” in the phrase “directory operation.” The unqualified term “operation” denotes the language-independent specification construct. Sometimes, to reinforce the distinction, the phrase “interface operation” is used to denote the language-independent specification construct, instead of “operation.”

The reader is urged to be cautious, and reference to 2.2 (Definitions) may be useful.

Language-independent concrete OM class names, OM attribute names, and OM attribute value names appear in bold font while abstract OM class names appear in bold italic font. They are all spelled with hyphens between words. The first letters of language-independent OM class and OM attribute names are capitalized (e.g., **Filter-Item**).

Language-independent datatype, operation argument, and error names appear in Helvetica font, are lowercased, and are spelled with underscores between words (e.g., `ds_abandon`).

Each OM class has an associated datatype, whose name is obtained from the class name by converting uppercase to lowercase, converting hyphens to underscores, adding the prefix “ds_”, and adding the suffix “_type”. (Thus, `ds_filter_item_type` is the datatype corresponding to OM class **Filter-Item**.)

When important terms are introduced, they appear in italics. Italics are also used in 2.2 for cross-references.

The use of fonts in this standard is as follows:

(1) The Helvetica font is used for

- Language-independent operation names, such as `ds_add_entry`
- Language-independent datatype names, such as `ds_status_type`
- Language-independent error names, such as `Abandon-Failed`

(2) The *italic* font is used for

- Language-independent operation arguments, such as *session*
- The introduction of important terms
- Cross-references in 2.2

(3) The **bold** font is used for

- Language-independent concrete OM class names, such as **Filter-Item**
- Language-independent OM attribute names, such as **Filter-Item-Type**
- Language-independent OM attribute values, such as **approximate-match**

(4) The ***bold italic*** font is used for

- Language-independent abstract OM class names, such as ***Common-Results***

(5) The **constant width** (Courier) font is used for

- Terms defined in standards upon which this standard is based

2.1.2 Representation of Strings of Numbers

Strings of numbers (such as ASN.1 Object Identifier encodings) are represented in the form “\xaa\xbb\xcc . . .”, where each term \xnn represents an element of the string, and the value of that element expressed in hexadecimal notation is nn. So, for example, the string consisting of the three numbers 1, 10, 100 is represented as “\x01\x0a\x64”.

2.1.3 Language-Independent Conventions

The language-independent specification conventions assume a model of an abstract OM object class mechanism and an abstract procedural interface, although programming language bindings to this interface may take other forms.

In the OM class definitions, each OM class is described in a separate subclause, 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. In each subsection describing an OM class, the OM attributes specific to that OM class are defined in a table. This 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` operation supplies. The class definitions include OM enumeration syntax definitions; the values of each OM enumeration syntax are listed where that syntax first appears as the syntax of an OM attribute. The class definitions also include definitions of constant values of OM attributes.

For the abstract procedural interface, the main elements of the notation are abstract datatype definitions, constant data value definitions and operation definitions. The conventions for these are described in IEEE Std 1224-1993 {14}.

2.2 Definitions

2.2.1 Terminology

For the purposes of this standard, the following definitions apply.

2.2.1.1 conformance document: A document provided by an implementor that contains implementation details as described in 1.3.1.2. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.2 implementation defined: An indication that the implementation shall define and document the requirements for correct program constructs and correct data of a value or behavior. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.3 locally administered: An indication that the configuration is not consistent across all systems and that the administrator of each system shall provide documentation of its behavior.

2.2.1.4 may: An indication of an optional feature.

With respect to implementations, the word *may* is to be interpreted as an optional feature that is not required in this standard, but can be provided. With respect to

Strictly Conforming Applications, the word *may* means that the optional feature shall not be used. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.5 shall: An indication of a requirement on the implementation or on strictly conforming applications, where appropriate. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.6 should:

(1) With respect to implementations, an indication of an implementation recommendation, but not a requirement.

(2) With respect to applications, an indication of a recommended programming practice for applications and a requirement for Strictly Conforming Applications. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.7 supported: A condition regarding optional functionality.

Certain functionality in this standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified in this standard (except that they do not return the error condition indicated for the unsupported case). If the functionality is not *supported*, the interface shall always return the indication specified for this situation. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.8 system documentation: All documentation provided with an implementation, except the conformance document.

Electronically distributed documents for an implementation are considered part of the system documentation. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.9 undefined: An indication that this standard imposes no portability requirements on an application's use of an indeterminate value or its behavior with erroneous program construct, or erroneous data.

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application using such behaviors is using extensions, as defined in 1.3.2.3. [ISO/IEC 9945-1: 1990 {B2}]

2.2.1.10 unspecified: An indication that this standard imposes no portability requirements on applications for correct program constructs or correct data regarding a value or behavior.

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions, as defined in 1.3.2.3. [ISO/IEC 9945-1: 1990 {B2}]

2.2.2 General Terms

For the purposes of this standard, the following definitions apply.

2.2.2.1 abstract class: An OM class of OM objects of which instances are forbidden.

2.2.2.2 Abstract Syntax Notation One (ASN.1): A notation that both enables complicated types to be defined and also enables values of these types to be specified (see ISO/IEC 8824: 1990 {1}).

2.2.2.3 access point: The point at which an abstract service is obtained. [ISO/IEC 9594-2: 1990 {6}]

2.2.2.4 address: An unambiguous name, label, or number that identifies the location of a particular entity or service.

See also *presentation address* (2.2.2.66).

2.2.2.5 alias [alias name]: An alternate name for a directory object, provided by the use of one or more alias entries in the DIT. [ISO/IEC 9594-2: 1990 {6}]

2.2.2.6 alias entry: A Directory entry, of Object Class “alias,” containing information used to provide an alternative name for an object.

2.2.2.7 alias name: See *alias* in 2.2.2.5.

2.2.2.8 argument: Information that is passed to an interface operation or a directory operation.

2.2.2.9 asynchronous completion: Completion of an asynchronous directory operation.

An asynchronous directory operation is complete when a corresponding synchronous directory operation would complete and any associated status fields have been updated.

2.2.2.10 asynchronous operation: An operation that does not of 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.

2.2.2.11 attribute: Used alone, means *directory attribute* as in 2.2.2.28 (see also *OM attribute*, 2.2.2.57).

2.2.2.12 attribute syntax [directory syntax]: A definition of the set of values that a directory attribute may assume.

It includes the datatype, in ASN.1, and, usually, one or more matching rules by which values may be compared.

2.2.2.13 attribute type: Used alone, means *directory attribute type* as in 2.2.2.29 (see also *OM attribute type*, 2.2.2.58).

2.2.2.14 attribute value: Used alone, means *directory attribute value* as in 2.2.2.30 (see also *OM attribute value*, 2.2.2.59).

2.2.2.15 attribute value assertion: A proposition, which may be true, false, or undefined, about the values (or perhaps only the distinguished values) of an entry. [ISO/IEC 9594-2: 1990 {6}]

NOTE: An attribute value assertion consists of an attribute type and a single value. Loosely, it is true if one of the values of the given attribute in the entry matches the given value.

2.2.2.16 attribute value syntax: See *OM syntax* in 2.2.2.62.

2.2.2.17 basic encoding rules: A specific set of rules used to encode ASN.1 values as strings of octets.

2.2.2.18 cache: See *copy* in 2.2.2.23.

2.2.2.19 certificate: A synonym for *user certificate*, 2.2.2.85.

2.2.2.20 chaining: A mode of interaction optionally used by a DSA that cannot perform a directory operation itself; the DSA chains by invoking a directory operation of another DSA and then relaying the outcome to the original requester. [ISO/IEC 9594-4: 1990 {8}]

2.2.2.21 class: See *object class* in 2.2.2.54 and *OM class* in 2.2.2.60.

2.2.2.22 continuation reference: A reference that describes how the performance of all or part of a directory operation can be continued at a different DSA or DSAs.

See also *referral* (2.2.2.71).

2.2.2.23 copy:

(1) A copy of an entry stored in other DSA(s) through bilateral agreement.

(2) A locally and dynamically stored copy of an entry resulting from a request (a cache copy).

- 1 **2.2.2.24 concrete class:** An OM class of which instances are permitted.
- 2 **2.2.2.25 descriptor:** The means by which the client and service exchange an OM
3 attribute value, and the integers that denote its representation, type, and syntax.
- 4 **2.2.2.26 descriptor list:** An ordered sequence of descriptors that is used to rep-
5 resent several OM attribute types and values.
- 6 **2.2.2.27 directory:** A repository of information about objects that provides direc-
7 tory services to its users, which allows its users access to the information.
- 8 **2.2.2.28 directory attribute [attribute]:** The information of a particular type
9 concerning an object and appearing in an entry describing the object in the DIB.
10 [ISO/IEC 9594-2: 1990 {6}]
- 11 **2.2.2.29 directory attribute type [attribute type]:** That component of a *direc-*
12 *tory attribute* that indicates the class of information given by that attribute.
13 It is an *object identifier*, and so completely unique. [ISO/IEC 9594-2: 1990 {6}]
- 14 **2.2.2.30 directory attribute value [attribute value]:** A particular instance of
15 the class of information indicated by a directory attribute type. [ISO/IEC 9594-2:
16 1990 {6}]
- 17 **2.2.2.31 directory class:** See *object class* in 2.2.2.54.
- 18 **2.2.2.32 directory information base:** The complete set of information to which
19 the directory provides access and that includes all of the pieces of information that
20 can be read or manipulated using the operations of the directory. [ISO/IEC 9594-
21 2: 1990 {6}]
- 22 **2.2.2.33 directory information tree:** The directory information base, consid-
23 ered as a tree, whose vertices (other than the root) are the directory entries.
24 [ISO/IEC 9594-2: 1990 {6}]
- 25 **2.2.2.34 directory object:** Anything in some “world,” generally the world of
26 telecommunications and information processing or some part thereof, that is
27 identifiable (can be named) and that it is of interest to hold information on in the
28 directory information base. [ISO/IEC 9594-2: 1990 {6}]
- 29 **2.2.2.35 directory operation:** Processing performed within the directory to pro-
30 vide a service, such as a read directory operation; it is given some arguments as
31 input, performs some processing, and returns some results.

1 An application process causes a directory operation to be performed by invoking an
2 interface operation.

3 **2.2.2.36 directory syntax:** See *attribute syntax* in 2.2.2.12.

4 **2.2.2.37 directory system agent:** An OSI application process that is part of the
5 Directory. [ISO/IEC 9594-2: 1990 {6}]

6 **2.2.2.38 directory user agent:** An OSI application process that represents a
7 user accessing the Directory. [ISO/IEC 9594-2: 1990 {6}]

8 NOTE: This may be composed of an arbitrary number of system processes and application processes,
9 including the one or more that are the user(s).

10 **2.2.2.39 distinguished encoding:** Restrictions to the basic encoding rules
11 designed to ensure a unique encoding of each ASN.1 value, defined in clause 8.7 of
12 ISO/IEC 9594-8: 1990 {11}.

13 **2.2.2.40 distinguished name:** One of the names of an object, formed from the
14 sequence of RDNs of its object entry and each of its superior entries. [ISO/IEC
15 9594-2: 1990 {6}]

16 **2.2.2.41 distinguished value:** An attribute value in an entry that has been
17 designated to appear in the relative distinguished name of the entry.

18 **2.2.2.42 entry [object entry]:** A part of the DIB that contains information
19 about an object.

20 Each entry is made up of attributes. [ISO/IEC 9594-2: 1990 {6}]

21 **2.2.2.43 filter:** An assertion about the presence or value of certain attributes of
22 an entry in order to limit the scope of a search. [ISO/IEC 9594-3: 1990 {7}]

23 **2.2.2.44 function:** A programming language construct, modeled after the mathe-
24 matical concept.

25 A function encapsulates some behavior. It is given zero or more arguments as
26 input, performs some processing, and returns some results. Functions are also
27 known as procedures, subprograms, or subroutines (see also *operation*, 2.2.2.63).

28 **2.2.2.45 immediate subordinate:** An entry in the DIT that is immediately
29 below another entry in the tree.

30 The distinguished name of the immediate subordinate is formed by appending its
31 relative distinguished name to the distinguished name of the other entry.

2.2.2.46 immediate superior: An entry in the DIT that is immediately above another entry in the tree.

The distinguished name of the immediate superior, followed by the relative distinguished name of the other entry, forms the distinguished name of the other entry.

Each entry (except the root) has exactly one immediate superior.

2.2.2.47 interface operation: See *operation* in 2.2.2.63.

2.2.2.48 invoke ID: An identifier used to distinguish one directory operation from all other outstanding operations.

2.2.2.49 knowledge reference: Knowledge that associates, either directly or indirectly, a DIT entry with the DSA in which it is located. [ISO/IEC 9594-4: 1990 {8}]

2.2.2.50 leaf entry: A Directory entry that has no subordinates.

It can be an alias entry or an object entry.

2.2.2.51 name: A construct that singles out a particular (directory) object from all other objects.

A name shall be unambiguous (that is, denote just one object); however, it need not be unique (that is, be the only name that unambiguously denotes the object). [ISO/IEC 9594-2: 1990 {6}]

2.2.2.52 nonspecific subordinate reference: A knowledge reference that holds information about a DSA that holds one or more unspecified subordinate entries. [ISO/IEC 9594-4: 1990 {8}]

2.2.2.53 object: See *directory object* in 2.2.2.34 and *OM object* in 2.2.2.61.

2.2.2.54 object class [directory class]: An identified family of objects (or conceivable objects) that share certain characteristics.

2.2.2.55 object entry: See *entry* in 2.2.2.42.

2.2.2.56 object identifier: A value (distinguishable from all other such values) that is associated with an information object (see ISO/IEC 8824: 1990 {1}).

2.2.2.57 OM attribute: A component of an OM object, comprised of an integer denoting the type of the attribute and an ordered sequence of one or more attribute values, each accompanied by an integer denoting the syntax of the value.

2.2.2.58 OM attribute type: A category into which OM attribute values are placed on the basis of their purpose.

2.2.2.59 OM attribute value: An arbitrarily complex information item that can be viewed as a characteristic or property of an OM object.

2.2.2.60 OM class: A category into which OM objects are placed on the basis of both their purpose and their internal structure.

2.2.2.61 OM object: Any of the complex information objects created, examined, modified, or destroyed by means of the interface.

2.2.2.62 OM syntax [attribute value syntax]: A category into which an OM attribute value is placed on the basis of its form.

2.2.2.63 operation: A program abstraction with formal input and output parameters that are bound to objects or values in the application that invokes it. [IEEE Std 1224-1993 {14}]

2.2.2.64 outstanding directory operation: A directory operation, invoked asynchronously (i.e., with **asynchronous** = **true** in the context), that has not yet been the subject of an invocation of the `ds_abandon` interface operation or the `ds_receive_result` interface operation.

2.2.2.65 package: A group of related OM classes.

2.2.2.66 presentation address: An unambiguous name that is used to identify a set of presentation service access points.

Loosely, it is the network address of an OSI service.

2.2.2.67 private OM object: The internal representation of the language-independent model of an OM object in a service, and thus unspecified. [IEEE Std 1224-1993 {14}]

2.2.2.68 process: An address space, a single thread of control that executes within that address space, and its required system resources.

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. [ISO/IEC 9945-1: 1990 {B2}]

NOTE: The term *process* is used in contrast to “system process,” or the OSI usage of the term “application process.”

- 1 **2.2.2.69 public OM object:** The representation by the client of an object in a
2 particular programming language binding. [IEEE Std 1224-1993 {14}]
- 3 **2.2.2.70 purported name:** A construct that is syntactically a name but that has
4 not (yet) been shown to be a valid name. [ISO/IEC 9594-2: 1990 {6}]
- 5 **2.2.2.71 referral:** An outcome that can be returned by a DSA that cannot per-
6 form a directory operation itself, and that identifies one or more other DSAs more
7 able to perform the directory operation. [ISO/IEC 9594-4: 1990 {8}]
- 8 **2.2.2.72 relative distinguished name:** A set of AVAs, each of which is true,
9 concerning the distinguished values of a particular entry. [ISO/IEC 9594-2: 1990
10 {6}]
- 11 **2.2.2.73 replication:** The process by which copies of entries are created and
12 maintained.
- 13 **2.2.2.74 result:** Information that is returned from an interface operation or a
14 directory operation and that constitutes the outcome of the processing that was
15 performed.
- 16 **2.2.2.75 schema:** The set of rules and constraints concerning DIT structure,
17 object class definitions, attribute types, and syntaxes that characterize the DIB.
18 [ISO/IEC 9594-2: 1990 {6}]
- 19 **2.2.2.76 service controls:** Parameters conveyed as part of a directory operation
20 that constrain various aspects of its performance. [ISO/IEC 9594-3: 1990 {7}]
- 21 **2.2.2.77 session:** A sequence of directory operations requested by a particular
22 user of a particular DUA using the same session OM object.
- 23 **2.2.2.78 signed:** The condition of information that has an enciphered summary
24 appended to it that is used to ensure the integrity of the data, the authenticity of
25 the originator, and the unambiguous relationship between the originator and the
26 data.
- 27 **2.2.2.79 subordinate:** The converse of superior. [ISO/IEC 9594-2: 1990 {6}]
- 28 **2.2.2.80 superior** (applying to entry or directory object): Immediately superior,
29 or superior to one that is immediately superior (recursively). [ISO/IEC 9594-2:
30 1990 {6}]
- 31 **2.2.2.81 synchronous operation:** An operation that causes the process request-
32 ing the operation to be blocked from further use of the CPU until it is complete.

- 1 **2.2.2.82 syntax:** See *attribute syntax* in 2.2.2.12 and *OM syntax* in 2.2.2.62.
- 2 **2.2.2.83 thread:** A single sequential flow of control within a process.
- 3 **2.2.2.84 user:** The entity or person that accesses the directory, used in this stan-
4 dard to refer to the application program that is using the interface.
- 5 **2.2.2.85 user certificate [certificate]:** The public keys of a user, together with
6 some other information, rendered unforgeable by encipherment with the secret
7 key of the certification authority that issued it. [ISO/IEC 9594-8: 1990 {11}]
- 8 **2.2.2.86 value:** See *attribute value* in 2.2.2.14 and *OM attribute value* in 2.2.2.59.
- 9 **2.2.2.87 workspace:** A repository for instances of OM classes in the closures of
10 one or more packages associated with the workspace.
- 11 **2.2.3 Abbreviations**
- 12 For the purposes of this standard, the following abbreviations apply.
- 13 **2.2.3.1 API:** Application Program Interface.
- 14 **2.2.3.2 ASN.1:** Abstract Syntax Notation One.
- 15 **2.2.3.3 AVA:** Attribute Value Assertion.
- 16 **2.2.3.4 BDCP:** Basic Directory Contents Package.
- 17 **2.2.3.5 BER:** Basic Encoding Rules.
- 18 **2.2.3.6 CA:** Certification Authority.
- 19 **2.2.3.7 CCITT:** International Telegraph and Telephone Consultative Committee.
- 20 **2.2.3.8 DAP:** Directory Access Protocol.
- 21 **2.2.3.9 DIB:** Directory Information Base.

- 1 **2.2.3.10 DIT:** Directory Information Tree.
- 2 **2.2.3.11 DL:** Delivery List.
- 3 **2.2.3.12 DMD:** Directory Management Domain.
- 4 **2.2.3.13 DN:** Distinguished Name.
- 5 **2.2.3.14 DS:** Directory Service.
- 6 **2.2.3.15 DS API:** Directory Services API (as defined in this standard).
- 7 **2.2.3.16 DS Package:** Directory Service Package.
- 8 **2.2.3.17 DSA:** Directory System Agent.
- 9 **2.2.3.18 DSP:** Directory System Protocol.
- 10 **2.2.3.19 DUA:** Directory User Agent.
- 11 **2.2.3.20 IA5:** International Alphabet No. 5.
- 12 **2.2.3.21 ID:** Identifier.
- 13 **2.2.3.22 the Interface:** The interface to directory services defined in this
14 standard.
- 15 **2.2.3.23 ISDN:** Integrated Services Digital Network.
- 16 **2.2.3.24 ISO:** International Organization for Standardization.
- 17 **2.2.3.25 ISO/IEC 9945-1:** ISO/IEC 9945-1: 1990 {B2}.
- 18 **2.2.3.26 MDUP:** MHS Directory User Package.

- 1 **2.2.3.27 MHS:** Message Handling System (as in ISO/IEC 10021: 1990 {B3}).
- 2 **2.2.3.28 MS:** Message Store.
- 3 **2.2.3.29 NBP:** Nonbasic Parameter.
- 4 **2.2.3.30 O/R:** Originator/Recipient.
- 5 **2.2.3.31 OM:** OSI abstract data Manipulation.
- 6 **2.2.3.32 OSI:** Open Systems Interconnection.
- 7 **2.2.3.33 POSIX.3:** IEEE Std 1003.3-1991 {B12}.
- 8 **2.2.3.34 RDN:** Relative Distinguished Name.
- 9 **2.2.3.35 ROSE:** Remote Operations Service Element (see ISO/IEC 9072-1: 1989
10 {3}).
- 11 **2.2.3.36 SAP:** Strong Authentication Package.
- 12 **2.2.3.37 X.500 directory standards:** ISO/IEC 9594: 1990 and the 1988 CCITT
13 X.500 Series Recommendations ({5–11}).

Section 3: Description

3.1 General

The interface contains a number of operations together with many OM classes of OM objects, which are used as the arguments and results of the interface operations.

NOTE: Both the operations and the OM objects are based closely on the *abstract service* that is specified in the X.500 directory standards (ISO/IEC 9594-3: 1990 {7}).

The interface models directory interactions as service requests made through a number of interface *operations*, which take a number of input *parameters*. Each valid request shall cause a *directory operation* within the directory, which eventually returns a *status* and any *result* of the directory operation.

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

The other arguments to the interface operations shall include a *context* and various service-specific arguments. The context includes a number of arguments that are common to many interface operations and that seldom change from one directory operation to another.

The X.500 directory standards define abstract services that requesters use to interact with the directory. Each of these abstract services shall map to a single interface operation as shown in the following table.

NOTE: The detailed specifications are given in Section 4.

Service	Operation
DirectoryBind	ds_bind
DirectoryUnbind	ds_unbind
Read	ds_read
Compare	ds_compare
Abandon	ds_abandon
List	ds_list
Search	ds_search
AddEntry	ds_add_entry
RemoveEntry	ds_remove_entry
ModifyEntry	ds_modify_entry
ModifyRDN	ds_modify_rdn

There are some interface operations that have no counterparts in the abstract service. Interface operation `ds_receive_result` is used with asynchronous directory operations, as explained in 3.7. Operations `ds_initialize`, `ds_shutdown`, and `ds_version` are used as described in 3.2.

The interface operations are those summarized in Table 3-1. Those that can execute asynchronously (i.e., invoke asynchronous directory operations) are indicated by an “a” in the table. All other interface operations shall always execute synchronously.

Table 3-1 – Interface Operations Summary

Name	Description
<code>ds_abandon</code>	Abandon the result of a pending, asynchronous directory operation
<i>a</i> <code>ds_add_entry</code>	Add a leaf entry to the directory information tree
<code>ds_bind</code>	Open a session with a directory user agent
<i>a</i> <code>ds_compare</code>	Compare a purported attribute value with the attribute value stored in the directory for a particular entry
<code>ds_initialize</code>	Initialize the interface and allocate a workspace
<i>a</i> <code>ds_list</code>	Enumerate the immediate subordinates of a particular directory entry
<i>a</i> <code>ds_modify_entry</code>	Perform an atomic modification of a directory entry
<i>a</i> <code>ds_modify_rdn</code>	Change the RDN of a leaf entry
<i>a</i> <code>ds_read</code>	Query information on a directory entry by name
<code>ds_receive_result</code>	Retrieve the result of an asynchronous directory operation
<i>a</i> <code>ds_remove_entry</code>	Remove a leaf entry from the directory information tree
<i>a</i> <code>ds_search</code>	Find entries of interest in a portion of the directory information tree
<code>ds_shutdown</code>	Discard a workspace
<code>ds_unbind</code>	Terminate a directory session
<code>ds_version</code>	Negotiate features of the interface and service

3.2 Initialization and Shutdown

The interface has an initialization and shutdown sequence that permits the negotiation of optional features. This involves the operations `ds_initialize`, `ds_version`, and `ds_shutdown`.

Every application program shall first invoke `ds_initialize`, which shall return a workspace that supports the standard Directory Service package (see Section 5). If this workspace supports other packages or extensions, then those packages or extensions shall not affect the behavior of applications that do not use them. Using the workspace as such means that any attribute values returned by the service shall be represented as OM syntaxes corresponding to ASN.1 simple types or as objects of OM class **Encoding**, as discussed in 3.5.1.

The workspace can be extended to support other packages such as the Basic Directory Contents package, the Strong Authentication package and the MHS Directory User package described in Section 7 or other extensions not described in this standard. All such packages or other extensions shall be identified by means of

OSI object identifiers, and these object identifiers shall be supplied to the `ds_version` operation to incorporate the extensions into the workspace.

NOTE: 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 Section 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 operations and can start one or more directory sessions using `ds_bind`.

Eventually—when it has completed its tasks, terminated all its directory sessions using `ds_unbind`, and released all its OM objects using `om_delete`—the application should ensure that resources associated with the workspace are freed by invoking `ds_shutdown`. This shall have the effect of deleting any private objects or service-generated public objects that are in the workspace.

It shall be possible for an application to create more than one workspace using `ds_initialize`. Each such workspace shall be distinct.

NOTE: This is to allow several uncoordinated activities, or libraries, to use these services without interfering with each other.

3.3 Session

A session identifies to which DUA and to which suite of DSAs a particular directory operation is to be sent. It contains some **Directory-BindArguments**, such as the distinguished name of the requester. It is passed as the first argument to most interface operations.

A session shall be described by an OM object of OM class **Session**. It shall be created, and appropriate parameter values may be set, using the OM operations. A directory session shall be started by invoking the operation `ds_bind` and shall later be terminated by invoking the operation `ds_unbind`. A session with default parameters can be started by passing the choice value **Default-Session** as the *session* parameter to `ds_bind`.

The `ds_bind` operation shall be invoked before the *session* is used as an argument to any other operation in this interface. After `ds_unbind` has been invoked, `ds_bind` shall be invoked again if another session is to be started.

The interface shall support multiple concurrent sessions.

NOTES:

- (1) This means 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; for example, when using threads.
- (2) It also means that a process can interact directly and concurrently with different parts of the directory.
- (3) Detailed specifications of the OM class **Session** are given in Section 5.

3.4 Context

A context defines the characteristics of the directory interaction that are specific to a particular directory operation, but often are used unchanged for many directory operations. Since these parameters are presumed to be relatively static for a given directory user during a particular directory interaction, these parameters 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 interface operation.

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. The **ServiceControls** include **options** (such as **preferChaining**, **chainingProhibited**, **localScope**, **dontUseCopy**, and **dontDereferenceAliases**) together with **priority**, **timeLimit**, **sizeLimit**, and **scopeOfReferral**. Each of these shall be mapped onto an OM attribute in the context, and they are detailed in Section 5. The effect shall be as though they were passed as a group of additional arguments on every operation invocation.

The value of each component of the context shall be determined when the interface operation is invoked, and it shall remain fixed throughout the directory operation.

The choice value **Default-Context** can be passed as the value of the *context* parameter to the interface operations. It shall have the same effect as a context OM object created with default values.

NOTE: All OM attributes in the *context* have default values, some of which are locally administered.

The context shall be a private object, unless it is **Default-Context**.

NOTE: Detailed specifications of the OM class **Context** are given in Section 5.

3.5 Operation Parameters

The abstract service defines specific parameters for each directory operation. These shall be mapped onto corresponding parameters to each interface operation (which are called input parameters).

NOTE: Although each service has different parameters, some specific parameters recur in several directory operations, and these are described here. Further details of these and all the other parameters are given in the operation definitions of Section 4 and the OM class definitions of Section 5.

All parameters that are OM objects can generally be supplied to the interface operations as public objects or as private objects. Private objects shall be created in a workspace that was returned by `ds_initialize`. In some cases, constants can be supplied instead of OM objects.

Wherever an operation is stated to accept an instance of a particular OM class as the value of a parameter, it shall also accept an instance of any subclass of the OM class.

NOTE: For example, most operations have a *name* parameter, which accepts values of OM class **Name**. It will always be valid to supply an instance of the subclass **DS-DN** as the value of the parameter.

When the application supplies a subclass of the specified OM class as a parameter, containing attributes not defined in the specified class, the service shall either recognize those attributes as extensions or shall ignore them.

3.5.1 Attribute and AVA

Each directory attribute shall be represented in the interface by an OM object of OM class **Attribute**. The type of the directory attribute shall be represented by an OM attribute, **Attribute-Type**, within the OM object.

The values of the directory attribute shall be expressed as the values of the OM attribute **Attribute-Values**. The representation of each attribute value shall depend on the attribute type and shall be determined as set out in the following list. This list shows the way in which an application program shall supply values to the interface (for example, in the *changes* parameter to the *ds_modify_entry* operation). The interface shall follow the same rules when returning attribute values to the application (for example, in the result of the *ds_read* operation).

- The first possibility is that the attribute type and the representation of the corresponding values are defined in a package, such as the selected attribute types from the X.500 directory standards that are defined in the Basic Directory Contents package in Section 7 of this standard. In this case, if use of the package has been negotiated by the application using the *ds_version* operation, attribute values shall be represented as specified in that package.

NOTE: Additional directory attribute types and their OM representations may be defined by future versions of this standard or by 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.

NOTE: These syntaxes are **String(Bit)**, **Boolean**, **Integer**, **Null**, **String(Object-Identifier)**, and **String(Octet)**. See IEEE Std 1224-1993 {14}.

In this case, the value shall be represented as 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 shall be represented as an object of OM class **Encoding**, as defined in IEEE Std 1224-1993 {14}.

NOTE: Where attribute values have OM syntax **String(*)**, they may be long, segmented strings, and the operations *om_read* and *om_write* should be used to access them.

1 An attribute-value assertion (AVA) is an assertion about the value of an attribute
2 of an entry, and can be true, false, or undefined. It consists of an attribute type
3 and a single value.

4 NOTE: Loosely, the AVA is true if one of the values of the given attribute in the entry matches the
5 given value.

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

8 Information used by `ds_add_entry` to construct a new directory entry shall be rep-
9 resented by an OM object of OM class **Attribute-List**, which contains a single,
10 multivalued OM attribute whose values are OM objects of OM class **Attribute**.

11 3.5.2 Entry Information Selection

12 The *selection* parameter of the `ds_read` and `ds_search` operations shall tailor their
13 results to obtain just part of the required entry. Information about all attributes,
14 no attributes, or just for a named set can be chosen. Attribute types shall always
15 be returned.

16 NOTE: The attribute values are not always returned.

17 Except where one of the constants below is used, the value of the parameter shall
18 be an instance of OM class **Entry-Information-Selection**. One of the constants
19 below can be used in simple cases.

20 — To verify the existence of an entry for the purported name, the constant
21 **Select-No-Attributes** can be used.

22 — To return just the types of all attributes, **Select-All-Types** can be used.

23 — To return the types and values of all attributes, **Select-All-Types-And-**
24 **Values** can be used.

25 To choose a particular set of attributes, the application shall create a new instance
26 of the OM class **Entry-Information-Selection** and set the appropriate OM attri-
27 bute values using the OM operations.

28 3.5.3 Name

29 Most directory operations take a *name* parameter to specify the target of the direc-
30 tory operation. The name shall be represented by an instance of one of the sub-
31 classes of the OM class **Name**.

32 For directory interrogations, any aliases in the name shall be dereferenced unless
33 prohibited by the **Dont-Dereference-Aliases** service control. For directory
34 modifications, any aliases in the name shall not be dereferenced.

35 NOTES:

36 (1) This standard defines the subclass **DS-DN** to represent distinguished names and other
37 names.

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

3 3.6 Operation Results

4 All interface operations return a *status*, most return an *invoke_id* that identifies a
5 particular directory operation invocation, and the interrogation interface opera-
6 tions each return a *result*.

7 All OM objects returned by interface operations (results and errors) shall be pri-
8 vate objects in the workspace associated with the directory session in the invoca-
9 tion of `ds_bind`.

10 3.6.1 The Invoke Id Parameter

11 Each interface operation that invokes a directory operation shall return an
12 *invoke_id*, which shall be a handle that identifies the particular invocation of a
13 directory operation. The *invoke_id* shall only be relevant for asynchronous direc-
14 tory operations and can be used later to receive the result and status or to aban-
15 don them.

16 NOTES:

- 17 (1) Asynchronous directory operations are fully described in 3.7.
18 (2) The interface operations that can be used to invoke asynchronous directory operations are
19 indicated by an “a” in Table 3-1.

20 The numerical value returned from an interface operation that successfully
21 invokes an asynchronous directory operation shall be unique among all outstand-
22 ing directory operations in a given session.

23 The value returned for a synchronous directory operation is unspecified, as is that
24 for an interface operation that fails to invoke a directory operation.

25 3.6.2 Result

26 Interrogation directory operations return a result only if they succeed. All errors
27 from these directory operations, including DAP errors, shall be reported in the *sta-*
28 *tus* described in 3.6.3, as shall errors from all other directory operations.

29 The value returned by an interface operation that invokes an asynchronous direc-
30 tory operation is unspecified, as is that for an interface operation that fails to
31 invoke a directory operation. The result of an asynchronous directory operation
32 shall be returned by a later invocation of `ds_receive_result`.

33 The result of an interrogation shall be returned in a private object whose OM class
34 is appropriate to the particular directory operation.

35 NOTE: The format of directory operation results is driven both by the abstract service and by the

need to provide asynchronous execution of interface operations. To simplify processing of asynchronous results, the result of a single directory operation is returned in a single OM object (corresponding to the abstract result defined in the X.500 directory standards).

The components of the result of a directory operation shall be represented by OM attributes in the result object of the directory operation. All information contained in the abstract service result shall be made available to the application program. The result can be inspected using the interface operations provided in the API defined in IEEE Std 1224-1993 {14}.

NOTE: Only the interrogation directory 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 Section 5. The results of the different directory operations share several common components, including the **CommonResults** defined in the X.500 directory standards (ISO/IEC 9594-3:1990 {7}), 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.

NOTE: Thus, the operation `om_instance` should always be used when testing the OM class.

Any attribute values in the result shall be represented as discussed in 3.5.1.

3.6.3 Status

Every interface operation shall return a *status* value, which shall be either a constant or an object.

If it is a constant, it shall be one of the following:

- The constant **success**
- The constant **no_workspace**
- The constant **invalid_workspace**

If it is an object, it shall be 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**)

NOTE: Details of all errors are given in Section 6.

Other results of operations shall not be treated as valid unless the status result has the value **success**.

3.7 Synchronous and Asynchronous Directory Operations

Asynchronous directory operations may be provided by the interface, determined for each directory operation by the value of the **Asynchronous** OM attribute in the *context* passed to the interface operation. The default value of this OM attribute shall be **false**, causing all directory operations to be synchronous. Asynchronous operation is an optional feature; directory operations shall always be synchronous on implementations that do not provide it. The value of **Max-Outstanding-Operations**, described below, shall indicate the presence of the feature.

In synchronous mode, each interface operation shall wait until the directory operation is complete before returning control to the application. Thus the thread of control shall be blocked within the interface after invoking an operation, and the application shall be able to make use of the result immediately after the operation finishes.

In asynchronous mode, some interface operations shall return control before the associated directory operations are complete. The application can then continue with other processing while the directory operation is being executed by the directory, and it can then access the result by invoking `ds_receive_result`. An application shall be able to initiate several concurrent asynchronous directory operations on the same session before receiving any of the results, subject to the limit described below. The application shall be able to cause an outstanding asynchronous directory operation to be abandoned by invoking `ds_abandon`.

NOTE: The interface operations that can invoke asynchronous directory operations are indicated in Table 3-1.

An interface operation invoking an asynchronous directory operation shall return an *invoke_id* of the directory operation to the application. The same *invoke_id* can be passed to `ds_abandon` to cause the directory operation to be abandoned, or it can be passed to `ds_receive_result` to receive the results for that directory operation.

Implementations of the interface are free to return control from asynchronous interface operations as soon as possible or may wait until the directory operation has been submitted to the underlying directory service. The policy followed is implementation defined.

Implementations shall also define a limit on the number of asynchronous directory operations that may be outstanding at any one time on any one session. An asynchronous directory operation is outstanding from the time that the interface operation is invoked until the result is returned by `ds_receive_result` or abandoned by `ds_abandon`. The limit shall be given by the constant **Max-Outstanding-Operations**, and it shall have the value zero if asynchronous operation is not supported. If the feature is present, it shall be at least one. While the maximum number of directory operations are outstanding, attempts to invoke further asynchronous directory operations shall report the error *too-many-operations* of class **Library-Error**.

A synchronous invocation (other than of `ds_abandon` or `ds_receive_result`) can return the mixed-synchronous error of class **Library-Error** if it is made on a session on which there are any outstanding asynchronous directory operations. All asynchronous directory operations should be allowed to terminate, and their results should be obtained, before making a synchronous invocation on the same session.

If an error is detected before an asynchronous request is submitted to the directory, the operation shall finish immediately and there shall be no outstanding directory operation generated. Errors that are detected after an asynchronous event is successfully submitted to the directory shall be reported to the application by `ds_receive_result` when the result of the outstanding asynchronous directory operation is returned. All errors occurring during a synchronous request shall be reported when the operation finishes.

NOTE: Full details of error handling are given in Section 6.

Where suitable system primitives⁴⁾ are provided, applications can obtain a file descriptor from the session by inspecting the value of the OM attribute **File-Descriptor**. Applications can 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 directory operations on a session when `ds_unbind` is invoked on that session.

NOTE: Once `ds_unbind` has been invoked, there is no way to determine whether any outstanding directory operations succeed or even whether they were ever sent to the directory. No errors or results of any kind will be reported to the application. It is strongly recommended that `ds_receive_result` is invoked repeatedly until **Completion-Flag** takes the value **No-Outstanding-Operation**.

3.8 Automatic Connection Management

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

NOTES:

- (1) Such management is intended to bring benefits such as reduced communication charges.
- (2) 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** directory operation specified in the X.500 directory standards be performed when the `ds_bind` interface operation is invoked.

4) Such as System V *poll()* (found in {B21}) or BSD (Berkeley Source Distribution) *select()* (found in {B22}). Future versions of ISO/IEC 9945-1:1990 {B2} may include such primitives.

3.9 Automatic Continuation and Referral Handling

By default, the interface shall provide automatic handling of continuation references and referrals.

NOTE: This is in order to reduce the burden on application programs.

A continuation reference describes how the performance of all or part of a directory operation can be continued at a different DSA or DSAs.

A single continuation reference, returned as the entire response to a directory operation, is called a *referral*. When a referral is returned, it is returned in the *status* of a *ds_list* or *ds_search* operation or in the *operation_status* parameter of *ds_receive_result* (in the case of a directory operation being invoked asynchronously).

One or more continuation references can also be returned as part of the response to a directory operation. In this case, they are returned in a **Partial-Outcome-Qualifier** attribute in the *result* parameter of a *ds_list* or *ds_search* interface operation or in the *result* parameter of *ds_receive_result* (in the case of an asynchronous directory operation).

NOTE: A DSA will return a referral if it has administrative, operational or technical reasons for preferring not to chain.

An implementation can return a referral if **Chaining-Prohibited** is set in the *context*, though it can instead report the chaining-required error of class **Service-Error** in this circumstance. Which of these courses it follows is unspecified.

By default, the implementation shall use any continuation references it receives to try to contact the other DSA(s), and so progress the directory operation, whenever practical. It shall only return the result or an error to the application after it has made this attempt.

NOTE: Continuation references may still be returned to the application if the relevant DSA cannot be contacted, for example.

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

Section 4: Interface Operations

4.1 Datatype Definitions

The following datatype definitions are used in operation definitions in this section.

NOTE: The definitions of other datatypes used in operation definitions in this section can be found in Sections 5 and 6.

Datatype: ds_status_type
Definition: choice [
 success : ds_null_type
 no_workspace : ds_null_type
 invalid_workspace : ds_null_type
 referral : ds_referral_type
 attribute_error : ds_attribute_error_type
 error : ds_error_type
]
Description: An instance of this datatype contains a procedure status return.

Datatype: ds_invoke_id_type
Definition: opaque
Description: An instance of this datatype contains an asynchronous directory operation invocation identifier.

Name: Any-Operation
Datatype: ds_invoke_id_type
Description: An instance of this datatype contains an unspecified invocation identifier.

Datatype: ds_null_type
Definition: state [null]
Description: This datatype is used where no data is present.

Datatype: ds_completion_flag_type
Definition: state [
 completed-operation,
 outstanding-operations,
 no-outstanding-operation,

1		other-completed-operations
2		]
3	Description:	An instance of this datatype contains an asynchronous directory
4		operations status.
5	Datatype:	ds_operation_result_type
6	Definition:	choice [
7		Null-Result: ds_null_type,
8		non_Null-Result: om_object_type
9		]
10	Description:	An instance of this datatype contains a result of an asynchro-
11		nous directory operation.
12	Datatype:	ds_subset_type
13	Definition:	state [
14		base-object,
15		one-level,
16		whole-subtree
17		]
18	Description:	An instance of this datatype contains a subset of a DIT to be
19		searched.
20	Datatype:	ds_feature_list_type
21	Definition:	list [om_object_id_type]
22	Description:	An instance of this datatype contains a list of candidate
23		features.
24	Datatype:	ds_activated_list_type
25	Definition:	list [boolean_type]
26	Description:	An instance of this datatype contains a list of activated features.

27 4.2 Abandon

28 4.2.1 Synopsis

29	Operation:	ds_abandon
30	Status Datatype:	ds_status_type
31	Input:	session : ds_session_type
32		invoke_id : ds_invoke_id_type

33 4.2.2 Description

34 This interface operation shall abandon a particular outstanding directory
 35 operation. The directory operation shall no longer be outstanding after
 36 ds_abandon finishes; therefore, its result shall not be returned by
 37 ds_receive_result.

1 NOTE: This interface operation may cause an **Abandon** directory operation to be requested of the
 2 directory. Such an **Abandon** directory operation may cause the directory to abandon the out-
 3 standing asynchronous directory operation itself.

4 4.2.3 Arguments In

5 *session* The directory session in which the directory operation was submitted
 6 to the directory. This shall be a private object.

7 *invoke_id*
 8 The invocation identifier of the specific outstanding asynchronous
 9 directory operation submitted via *session* that is to be terminated.
 10 The directory operation shall be an interrogation (i.e., invoked
 11 through *ds_compare*, *ds_list*, *ds_read*, or *ds_search*). The value of
 12 *invoke_id* shall be that returned by the interface operation that ini-
 13 tiated the asynchronous directory operation to be abandoned.

14 4.2.4 Results

15 *status* Whether the operation completed successfully.

16 4.2.5 Errors

17 This operation can return a error of class **System-Error** or one of the following
 18 errors of class **Library-Error**: bad-session, miscellaneous.

19 This operation can return an error of class **Communications-Error** or the follow-
 20 ing directory error: abandon-failed. The result of the asynchronous directory
 21 operation shall not be returned even if an abandon-failed error is returned.

22 This operation can return the error constant **no_workspace**.

23 4.3 Add Entry

24 4.3.1 Synopsis

25 **Operation:** *ds_add_entry*
 26 Status Datatype: *ds_status_type*
 27 Input: *session* : *ds_session_type*
 28 *context* : *ds_context_type*
 29 *name* : *ds_name_type*
 30 *entry* : *ds_attribute_list_type*
 31 Output: *invoke_id* : *ds_invoke_id_type*

4.3.2 Description

This operation shall add a leaf entry to the directory. The entry shall only be added if it conforms to the directory schema. The entry can be either an object or an alias.

4.3.3 Arguments In

session The directory session through which to submit the request. This shall be a private object.

context The directory context to use for the directory operation. **Size-Limit** and **Dont-Dereference-Aliases** shall not apply to the directory operation. This argument shall be a private object or the choice value **Default-Context**.

name The name of the entry to be added. The immediate superior of the new entry shall be determined by removing the last RDN component (which belongs to the new entry). The immediate superior should exist in the same DSA; otherwise, the operation can fail with the affects-multiple-dsas error of class **Update-Error**. Any aliases in the name shall not be dereferenced.

entry The attribute information that, together with the information from the RDN, constitutes the entry to be created.

NOTE: 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**.

4.3.4 Arguments Out

invoke_id The invocation identifier of an asynchronous directory operation.

4.3.5 Results

status Whether the entry was added (if used synchronously) or whether the directory operation was initiated (if used asynchronously).

4.3.6 Errors

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

The following directory errors can be returned by the operation (or by `ds_receive_result` when used asynchronously): `attribute-error`, `name-error`, `referral`, `security-error`, `service-error`, or `update-error`.

The `affects-multiple-dsas` error of class **Update-Error**, which is referred to in the argument descriptions, need not necessarily be returned if there is local agreement between the DSAs to allow the entry to be added.

This operation can return an error of class **Communications-Error**.

This operation can return the error constant **no_workspace**.

4.4 Bind

4.4.1 Synopsis

Operation: `ds_bind`
Status Datatype: `ds_status_type`
Input: `session` : `ds_session_type`
`workspace` : `om_workspace_type`
Output: `bound_session` : `ds_session_type`

4.4.2 Description

This operation shall open a session with the directory.

NOTE: In order to allow automatic connection management, `ds_bind` need not communicate with a DSA when it is invoked.

The interface shall support multiple concurrent sessions.

4.4.3 Arguments In

session A particular directory service provider, together with other details of the service required. Either a public object or a private object can be supplied as the value of this argument. The choice value **Default-Session** can also be used as the value of this argument. It shall cause a new session to be created with default values for all its OM attributes.

workspace The workspace (obtained from an invocation of `ds_initialize`) that is to be associated with the session. All interface operation results from directory operations using this session shall be returned as private objects in this workspace. If the *session* argument is a private object, it shall be a private object in this workspace.

4.4.4 Arguments Out

bound_session

Upon successful completion, this argument shall contain an instance of a directory session that can be used as an argument to other interface operations (e.g., *ds_read*). This shall be a new private object if the value of *session* was **default-session** or a public object; otherwise, it shall be the value of *session*. The interface operation shall supply default values for any of the OM attributes that were not present in the *session* instance supplied as an argument. It shall also set the value of the **File-Descriptor** OM attribute (the value shall be **No-Valid-File-Descriptor** if the functionality is not supported).

4.4.5 Results

status Whether the operation completed successfully.

4.4.6 Errors

This operation can return an error of class **System-Error** or one of the following errors of class **Library-Error**: bad-session, miscellaneous, or too-many-sessions.

The following directory errors can be returned by the operation: security-error or service-error.

This operation can return an error of class **Communications-Error**.

This operation can return the error constant **invalid_workspace**.

4.4.7 Cross-References

ds_unbind, 4.15.

4.5 Compare

4.5.1 Synopsis

Operation: *ds_compare*
Status Datatype: *ds_status_type*
Input: *session* : *ds_session_type*
context : *ds_context_type*
name : *ds_name_type*
ava : *ds_ava_type*
Output: *result* : *ds_compare_result_type*
invoke_id : *ds_invoke_id_type*