

**Information technology —
Test methods for measuring
conformance to directory
services —
Application Program Interface
(API) (Language independent)**

ICS 35.100.70

National foreword

This British Standard reproduces verbatim ISO/IEC 14393 : 1996, and implements it as the UK national standard.

The UK participation in its preparation was entrusted to Technical Committee IST/21, Open Systems Interconnection, Data Management and Open Distributed Processing, which has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee is available on request.

Cross-references

The British Standards which implement international or European publications referred to in this document may be found in the BSI Standards Catalogue under the section entitled 'International Standards Correspondence Index', or using the 'Find' facility of the BSI Standards Electronic Catalogue.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, the ISO/IEC front cover, pages ii to viii, pages 1 to 232, an inside back cover and a back cover.

This British Standard, having been prepared under the direction of the DISC Board, was published under the authority of the Standards Board and comes into effect on 15 November 1996

© BSI 1996

ISBN 0 580 26575 7

Amendments issued since publication

Amd. No.	Date	Text affected

INTERNATIONAL
STANDARD

**ISO/IEC
14393**

First edition
1996-06-01

**Information technology — Test methods
for measuring conformance to directory
services — Application Program Interface
(API) [Language independent]**

*Technologies de l'information — Méthodes d'essai pour mesurer la
conformité aux services de l'annuaire — Interface de programme
d'application (API) [indépendante du langage]*



Reference number
ISO/IEC 14393:1996(E)



Documents
Document Sharing Hub

Contents

	PAGE
Section 1: General	1
1.1 Scope	1
1.2 Normative References	2
1.3 Conformance	3
Section 2: Terminology and General Requirements	5
2.1 Conventions	5
2.2 Definitions	7
Section 3: Test Assertions—General	19
3.1 Scope	19
3.2 Normative References	19
3.3 Conformance	19
Section 4: Test Assertions—Terminology and General Requirements	21
Section 5: Test Assertions—Description	23
5.1 General	23
5.2 Initialization and Shutdown	23
5.3 Session	24
5.4 Context	24
5.5 Operation Parameters	24
5.6 Operation Results	27
5.7 Synchronous and Asynchronous Operations	28
5.8 Automatic Connection Management	30
5.9 Automatic Continuation and Referral Handling	30
Section 6: Test Assertions—Interface Operations	31
6.1 Datatype Definitions	31
6.2 Abandon	31
6.3 Add-Entry	35
6.4 Bind	47
6.5 Compare	53
6.6 Initialize	66

6.7	List	67
6.8	Modify-Entry	82
6.9	Modify-RDN	95
6.10	Read	108
6.11	Receive-Result	122
6.12	Remove-Entry	126
6.13	Search	137
6.14	Shutdown	155
6.15	Unbind	156
6.16	Version	159
Section 7: Test Assertions—Interface Class Definitions		161
7.1	Overview	161
7.2	Class Hierarchy	161
7.3	Access Point	161
7.4	Address	162
7.5	Attribute	162
7.6	Attribute List	163
7.7	AVA	163
7.8	Common Results	163
7.9	Compare Result	165
7.10	Context	166
7.11	Continuation Reference	169
7.12	DS DN	171
7.13	DS RDN	171
7.14	Entry Information	172
7.15	Entry Information Selection	173
7.16	Entry Modification	174
7.17	Entry Modification List	174
7.18	Extension	175
7.19	Filter	175
7.20	Filter Item	176
7.21	List Info	178
7.22	List Info Item	178
7.23	List Result	179
7.24	Name	180
7.25	Operation Progress	180
7.26	Partial Outcome Qualifier	181
7.27	Presentation Address	182
7.28	Read Result	183
7.29	Relative Name	183
7.30	Search Info	184
7.31	Search Result	184
7.32	Session	185
Section 8: Test assertions—Error Class Definitions		187
8.1	Introduction	187
8.2	OM Class Hierarchy	188
8.3	Error	188
8.4	Abandon Failed	189
8.5	Attribute Error	190

8.6	Attribute Problem	190
8.7	Communications Error	193
8.8	Library Error	194
8.9	Name Error	196
8.10	Referral	198
8.11	Security Error	198
8.12	Service Error	199
8.13	System Error	202
8.14	Update Error	203
Section 9: Test assertions—The X.500 Directory Service		205
9.1	Introduction	208
9.2	Package Object Identifiers	208
9.3	Selected Attribute Types	208
9.4	Selected Object Classes	214
9.5	OM Class Hierarchy	214
9.6	Algorithm Identifier	215
9.7	Certificate	215
9.8	Certificate List	216
9.9	Certificate Pair	216
9.10	Certificate Sublist	216
9.11	Certificates	217
9.12	Cross Certificates	217
9.13	DL Submit Permission	217
9.14	Facsimile Telephone Number	218
9.15	Forward Certification Path	218
9.16	Postal Address	218
9.17	Search Criterion	219
9.18	Search Guide	219
9.19	Signature	219
9.20	Teletex Terminal Identifier	220
9.21	Telex Number	220
Section 10: Test Assertions—Constant Definitions		221
10.1	Introduction	221
10.2	Directory Service Package	221
10.3	Basic Directory Contents Package	223
10.4	Strong Authentication Package	223
10.5	MHS Directory User Package	224
Annex A (informative) Bibliography		225
Alphabetic Topical Index		227

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 14393 was prepared by IEEE (as IEEE Std 1326.2-1993) and was adopted, under a special “fast-track procedure”, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Annex A of this International Standard is for information only.

1 Introduction

(This introduction is not a normative part of ISO/IEC 14393, Information technology—Test methods for measuring conformance to directory services—Application Program Interface (API) [Language independent], but is included for information only.)

The purpose of this International Standard is to define test methods for the language-independent specification contained in ISO/IEC 14392 {13} of the application program interface (API) to directory services.

A directory is a distributed collection of information that programs can access in order to make queries or updates. ISO/IEC 14392 {13} defines, in programming language-independent terms, an API to directory services. This API is known as the Directory Services API (DS API).

The DS API 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 CCITT X.500 Series recommendations and in ISO/IEC 9594. ISO/IEC 14392 {13} prescribes how the DS API is to be used to access the particular directory service defined in ISO/IEC 9594 and indicates how it may be used to access other directory services that conform to the same abstract model. Nothing in ISO/IEC 14392 {13} or in this International 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.

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 to access a single directory service or to access several directory services at the same time.

28 Related Standards

ISO/IEC 14392 {13} is intended to provide the basis for the definition of programming language bindings to which implementations and applications can conform. A specification for such a language binding, for the C programming language, is contained in ISO/IEC 14394 {B8}. This International Standard applies to test methods for measuring conformance to any such programming language binding specification.

The API defined in ISO/IEC 14392 {13} uses the mechanism for OSI abstract data Manipulation (OM) defined in ISO/IEC 14360 {12}. ISO/IEC 14362 {B5} defines the requirements that shall apply to test methods for measuring conformance to ISO/IEC 14360 {12}.

39 IEEE Std 1003.3-1991 {14} defines the general requirements that shall apply to
40 test methods for measuring conformance to POSIX. A set of test methods used to
41 measure conformance to ISO/IEC 14392 {13} must satisfy the requirements of
42 IEEE Std 1003.3-1991 {14}, as well as conforming to this International Standard.

43 **Overview**

44 For each section of ISO/IEC 14392 {13}, this International Standard contains a
45 corresponding section containing test assertions, in accordance with IEEE Std
46 1003.3-1991 {14}. A set of test methods conforms to this International Standard if
47 it tests all of the test assertions.

48 This International Standard is based on IEEE Std 1326.2-1993 {B9}, which was
49 prepared by the Namespace and Directory Services Working Group (P1224.2, form-
50 erly P1003.17), sponsored by the Portable Applications Standards Committee of
51 the IEEE Computer Society.

Information technology—Test methods for measuring conformance to directory services—Application Program Interface (API) [Language independent]

Section 1: General

1.1 Scope

This International Standard defines requirements for test methods for measuring conformance to ISO/IEC 14392 {13}.

ISO/IEC 14392 {13} is stated in terms that are independent of any particular programming language, but each implementation that conforms to it implements a particular programming language binding, and conforms to its programming language binding specification. Each set of test methods for measuring conformance to ISO/IEC 14392 {13} assumes a particular programming language binding specification. Each programming language binding specification may impose further, programming language specific, requirements on the test methods that, in conjunction with the requirements imposed by this International Standard, constitute the requirements that shall be satisfied by test methods used for measuring conformance to that programming language binding specification. This International Standard applies to all sets of test methods for measuring conformance to any programming language binding specification for ISO/IEC 14392 {13}.

1.2 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

- {1} ISO/IEC 8824: 1990 (CCITT Recommendation X.208: 1988),¹⁾ *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—Open Systems Interconnection—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—Open Systems Interconnection—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.*

1) ISO/IEC documents can be obtained from the ISO Central Secretariat, 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse. CCITT documents can be obtained from the Telecommunication Standardization Bureau of the International Telecommunication Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

- {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*.
- {12} ISO/IEC 14360: 1996, *Information technology—Open Systems Interconnection (OSI) abstract data manipulation—Application Program Interface (API) [Language independent]*.
- {13} ISO/IEC 14392: 1996, *Information technology—Directory services—Application Program Interface (API) [Language independent]*.
- {14} IEEE Std 1003.3-1991,²⁾ *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX*.

1.3 Conformance

A set of test methods that conforms to this International Standard shall conform to IEEE Std 1003.3-1991 {14}, with references in IEEE Std 1003.3-1991 {14} to the “POSIX.n test method specification” being interpreted as references to this International Standard, and references in IEEE Std 1003.3-1991 {14} to “the POSIX standard for which conformance is being measured” being interpreted as references to ISO/IEC 14392 {13}.

In addition to meeting the conformance criteria defined in IEEE Std 1003.3-1991 {14}, a set of test methods that conforms to this International Standard shall test all documentation assertions defined in this International Standard.

NOTE: Conformance to IEEE Std 1003.3-1991 {14} implies that the test methods will test all other assertions defined in this International Standard.

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

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 in describing both OSI abstract data manipulation and directory services. 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 International 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, by the language independent specification conventions and 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 International 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 International 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 conventions assume a model of an abstract OM object class mechanism and an abstract procedural interface, although programming

language bindings may take other forms. The language-independent specification conventions are described in ISO/IEC 14360 {12}.

2.2 Definitions

2.2.1 Terminology

For the purposes of this International 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 of ISO/IEC 9945-1 {B1}.

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 {B1}]

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. [ISO/IEC 14392 {13}]

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 International Standard, but that 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 {B1}]

2.2.1.5 shall: An indication of a requirement on the implementation or on Strictly Conforming Applications, where appropriate. [ISO/IEC 9945-1 {B1}]

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 {B1}]

2.2.1.7 supported: A condition regarding optional functionality.

Certain functionality in this International Standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified in this International 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 {B1}]

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 {B1}]

2.2.1.9 undefined: An indication that this International Standard imposes no portability requirements on an application's use of an indeterminate value or its behavior with erroneous program constructs 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 of ISO/IEC 9945-1 {B1}.

2.2.1.10 unspecified: An indication that this International 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 of ISO/IEC 9945-1 {B1}.

2.2.2 General Terms

For the purposes of this International Standard, the following definitions apply:

2.2.2.1 abstract class: An OM class of OM objects of which instances are forbidden. [ISO/IEC 14392 {13}]

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 {1}. [ISO/IEC 14392 {13}]

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

2.2.2.4 address: An unambiguous name, label, or number that identifies the location of a particular entity or service. [ISO/IEC 14392 {13}]

See also *presentation address* in 2.2.2.70.

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 {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. [ISO/IEC 14392 {13}]

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. [ISO/IEC 14392 {13}]

2.2.2.9 assertion: A statement that is derived from the standard to which conformance is being measured, that is true for a conforming implementation, and that pertains either to functionality or behavior of a functional interface or namespace allocation or to the documentation associated with the implementation being tested. [ISO/IEC 14360 {12}]

2.2.2.10 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. [ISO/IEC 14392 {13}]

2.2.2.11 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. [ISO/IEC 14392 {13}]

2.2.2.12 attribute: Used alone, means *directory attribute* as in 2.2.2.30 (but see also *OM attribute* in 2.2.2.61).

2.2.2.13 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. [ISO/IEC 14392 {13}]

2.2.2.14 attribute type: Used alone, means *directory attribute type* as in 2.2.2.31 (see also *OM attribute type* in 2.2.2.62).

2.2.2.15 attribute value: Used alone, means *directory attribute value* as in 2.2.2.32 (see also *OM attribute value* in 2.2.2.63).

2.2.2.16 attribute value assertion: A proposition, which may be true, false, or undefined, according to the specified matching rules for the type, concerning the presence in an entry of an attribute value (or a distinguished value) of a particular type. [ISO/IEC 9594-2 {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.

2.2.2.17 attribute value syntax: See *OM syntax* in 2.2.2.66.

2.2.2.18 base assertion: An assertion that is required to be tested for required features and for implemented conditional features. [IEEE Std 1003.3-1991 {14}]

2.2.2.19 basic encoding rules: A specific set of rules used to encode ASN.1 values as strings of octets. [ISO/IEC 14392 {13}]

2.2.2.20 cache: See *copy* in 2.2.2.25.

2.2.2.21 certificate: A synonym for *user certificate*, 2.2.2.89.

2.2.2.22 chaining: A mode of interaction optionally used by a DSA that cannot perform an 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 {8}]

2.2.2.23 class: See *object class* in 2.2.2.58 and *OM class* in 2.2.2.64.

2.2.2.24 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. [ISO/IEC 14392 {13}]

See also *referral* in 2.2.2.75.

2.2.2.25 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). [ISO/IEC 14392 {13}]

2.2.2.26 concrete class: An OM class of which instances are permitted. [ISO/IEC 14392 {13}]

2.2.2.27 descriptor: The means by which the client and service exchange an OM attribute value, and the integers that denote its representation, type, and syntax. [ISO/IEC 14392 {13}]

2.2.2.28 descriptor list: An ordered sequence of descriptors that is used to represent several OM attribute types and values. [ISO/IEC 14392 {13}]

2.2.2.29 directory: A repository of information about objects that provides directory services to its users, which allows its users access to the information. [ISO/IEC 9594-2 {6}]

2.2.2.30 directory attribute [attribute]: The information of a particular type concerning an object and appearing in an entry describing the object in the DIB. [ISO/IEC 9594-2 {6}]

2.2.2.31 directory attribute type [attribute type]: That component of a *directory attribute* that indicates the class of information given by that attribute.

It is an *object identifier*, and so completely unique. [ISO/IEC 9594-2 {6}]

2.2.2.32 directory attribute value [attribute value]: A particular instance of the class of information indicated by a directory attribute type. [ISO/IEC 9594-2 {6}]

2.2.2.33 directory class: See *object class* in 2.2.2.58.

2.2.2.34 directory information base: The complete set of information to which the directory provides access and that includes all of the pieces of information that can be read or manipulated using the operations of the directory. [ISO/IEC 9594-2 {6}]

2.2.2.35 directory information tree: The directory information base, considered as a tree, whose vertices (other than the root) are the directory entries. [ISO/IEC 9594-2 {6}]

2.2.2.36 directory object: Anything in some “world,” generally the world of telecommunications and information processing or some part thereof, that is identifiable (can be named) and that it is of interest to hold information on in the directory information base. [ISO/IEC 9594-2 {6}]

2.2.2.37 directory operation: Processing performed within the directory to provide a service, such as a read directory operation; it is given some arguments as input, performs some processing, and returns some results.

An application process causes a directory operation to be performed by invoking an interface operation. [ISO/IEC 14392 {13}]

2.2.2.38 directory syntax: See *attribute syntax* in 2.2.2.13.

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

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

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

2.2.2.41 distinguished encoding: Restrictions to the basic encoding rules designed to ensure a unique encoding of each ASN.1 value, defined in clause 8.7 of ISO/IEC 9594-8 {11}. [ISO/IEC 14392 {13}]

- 227 **2.2.2.42 distinguished name:** One of the names of an object, formed from the
 228 sequence of RDNs of its object entry and each of its superior entries. [ISO/IEC
 9594-2 {6}]
- 229 **2.2.2.43 distinguished value:** An attribute value in an entry that has been
 230 designated to appear in the relative distinguished name of the entry. [ISO/IEC
 231 14392 {13}]
- 232 **2.2.2.44 documentation assertion:** An assertion that pertains to the documen-
 233 tation associated with the implementation being tested. [ISO/IEC 14362 {B5}]
- 234 **2.2.2.45 entry [object entry]:** A part of the DIB that contains information about
 235 an object.
 236 Each entry is made up of attributes. [ISO/IEC 9594-2 {6}]
- 237 **2.2.2.46 extended assertion:** An assertion that is not required to be tested.
 238 [IEEE Std 1003.3-1991 {14}]
- 239 **2.2.2.47 filter:** An assertion about the presence or value of certain attributes of an
 240 entry in order to limit the scope of a search. [ISO/IEC 9594-3 {7}]
- 241 **2.2.2.48 function:** A programming language construct, modeled after the
 242 mathematical concept.
 243 A function encapsulates some behavior. It is given zero or more arguments as
 244 input, performs some processing, and returns some results. Functions are also
 245 known as procedures, subprograms, or subroutines. See also *operation*, 2.2.2.67.
 246 [ISO/IEC 14392 {13}]
- 247 **2.2.2.49 immediate subordinate:** An entry in the DIT that is immediately
 248 below another entry in the tree.
 249 The distinguished name of the immediate subordinate is formed by appending its
 250 relative distinguished name to the distinguished name of the other entry.
 251 [ISO/IEC 14392 {13}]
- 252 **2.2.2.50 immediate superior:** An entry in the DIT that is immediately above
 253 another entry in the tree.
 254 The distinguished name of the immediate superior, followed by the relative dis-
 255 tinguished name of the other entry, forms the distinguished name of the other
 256 entry.
 257 Each entry (except the root) has exactly one immediate superior. [ISO/IEC 14392
 258 {13}]
- 259 **2.2.2.51 interface operation:** See *operation* in 2.2.2.67.

- 260 **2.2.2.52 invoke ID:** An identifier used to distinguish one directory operation from
all other outstanding operations. [ISO/IEC 14392 {13}]
- 261 **2.2.2.53 knowledge reference:** Knowledge that associates, either directly or
262 indirectly, a DIT entry with the DSA in which it is located. [ISO/IEC 9594-4 {8}]
- 263 **2.2.2.54 leaf entry:** A directory entry that has no subordinates.
264 It can be an alias entry or an object entry. [ISO/IEC 14392 {13}]
- 265 **2.2.2.55 name:** A construct that singles out a particular (directory) object from all
266 other objects.
267 A name shall be unambiguous (that is, denote just one object); however, it need not
268 be unique (that is, be the only name that unambiguously denotes the object).
269 [ISO/IEC 9594-2 {6}]
- 270 **2.2.2.56 nonspecific subordinate reference:** A knowledge reference that holds
271 information about a DSA that holds one or more unspecified subordinate entries.
272 [ISO/IEC 9594-4 {8}]
- 273 **2.2.2.57 object:** See *directory object* in 2.2.2.36 and *OM object* in 2.2.2.65.
- 274 **2.2.2.58 object class [directory class]:** An identified family of objects (or con-
275 ceivable objects) that share certain characteristics. [ISO/IEC 14392 {13}]
- 276 **2.2.2.59 object entry:** See *entry* in 2.2.2.45.
- 277 **2.2.2.60 object identifier:** A value (distinguishable from all other such values)
278 that is associated with an information object. See ISO/IEC 8824 {1}. [ISO/IEC
279 14392 {13}]
- 280 **2.2.2.61 OM attribute:** A component of an OM object, comprised of an integer
281 denoting the type of the attribute and an ordered sequence of one or more attribute
282 values, each accompanied by an integer denoting the syntax of the value.
- 283 **2.2.2.62 OM attribute type:** A category into which OM attribute values are
284 placed on the basis of their purpose.
- 285 **2.2.2.63 OM attribute value:** An arbitrarily complex information item that can
286 be viewed as a characteristic or property of an OM object.
- 287 **2.2.2.64 OM class:** A category into which OM objects are placed on the basis of
288 both their purpose and their internal structure.
- 289 **2.2.2.65 OM object:** Any of the complex information objects created, examined,
290 modified, or destroyed by means of the OM interface.

- 291 **2.2.2.66 OM syntax [attribute value syntax]:** A category into which an OM
attribute value is placed on the basis of its form.
- 292 **2.2.2.67 operation:** A program abstraction with formal input and output param-
293 eters that are bound to objects or values in the application that invokes it.
294 [ISO/IEC 14360 {12}]
- 295 **2.2.2.68 outstanding directory operation:** A directory operation, invoked
296 asynchronously (i.e., with **asynchronous = true** in the context), that has not yet
297 been the subject of an invocation of the `ds_abandon` interface operation or the `ds_-`
298 `receive_results` interface operation. [ISO/IEC 14392 {13}]
- 299 **2.2.2.69 package:** A group of related OM classes.
- 300 **2.2.2.70 presentation address:** An unambiguous name that is used to identify a
301 set of presentation service access points.
302 Loosely, it is the network address of an OSI service. [ISO/IEC 14392 {13}]
- 303 **2.2.2.71 private OM object:** The internal representation of the language-
304 independent model of an *OM object* in a service, and thus unspecified. [ISO/IEC
305 14360 {12}]
- 306 **2.2.2.72 process:** An address space, a single thread of control that executes
307 within that address space, and its required system resources.
308 On a system that implements *threads*, a process is redefined to consist of an
309 address space with one or more threads executing within that address space and
310 their required system resources. [ISO/IEC 9945-1 {B1}]
- 311 NOTE: The term *process* is used in contrast to “system process,” or the OSI usage of the term “appli-
312 cation process.”
- 313 **2.2.2.73 public OM Object:** The representation by the client of an object in a
314 particular programming language binding. [ISO/IEC 14360 {12}]
- 315 **2.2.2.74 purported name:** A construct that is syntactically a name but that has
316 not (yet) been shown to be a valid name. [ISO/IEC 9594-2 {6}]
- 317 **2.2.2.75 referral:** An outcome that can be returned by a DSA that cannot per-
318 form a directory operation itself, and that identifies one or more other DSAs more
319 able to perform the directory operation. [ISO/IEC 9594-4 {8}]
- 320 **2.2.2.76 relative distinguished name:** A set of AVAs, each of which is true, con-
321 cerning the distinguished values of a particular entry. [ISO/IEC 9594-2 {6}]
- 322 **2.2.2.77 replication:** The process by which copies of entries are created and
323 maintained. [ISO/IEC 14392 {13}]

- 324 **2.2.2.78 result:** Information that is returned from an interface operation or a
325 directory operation and that constitutes the outcome of the processing that was
performed. [ISO/IEC 14392 {13}]
- 326 **2.2.2.79 schema:** The set of rules and constraints concerning DIT structure,
327 object class definitions, attribute types, and syntaxes that characterize the DIB.
328 [ISO/IEC 9594-2 {6}]
- 329 **2.2.2.80 service controls:** Parameters conveyed as part of a directory operation
330 that constrain various aspects of its performance. [ISO/IEC 9594-3 {7}]
- 331 **2.2.2.81 session:** A sequence of directory operations requested by a particular
332 user of a particular DUA using the same session OM object. [ISO/IEC 14392 {13}]
- 333 **2.2.2.82 signed:** The condition of information that has an enciphered summary
334 appended to it that is used to ensure the integrity of the data, the authenticity of
335 the originator, and the unambiguous relationship between the originator and the
336 data. [ISO/IEC 14392 {13}]
- 337 **2.2.2.83 subordinate:** The converse of superior. [ISO/IEC 9594-2 {6}]
- 338 **2.2.2.84 superior** (applying to entry or directory object): Immediately superior,
339 or superior to one that is immediately superior (recursively). [ISO/IEC 9594-2 {6}]
- 340 **2.2.2.85 syntax:** See *attribute syntax* in 2.2.2.13 and *OM syntax* in 2.2.2.66.
- 341 **2.2.2.86 test methods:** The software, procedures, or other means specified to
342 measure conformance to a specification. [ISO/IEC 14360 {12}]
- 343 **2.2.2.87 thread:** A single sequential flow of control within a process. [ISO/IEC
344 14392 {13}]
- 345 **2.2.2.88 user:** The entity or person that accesses the Directory, used in this Inter-
346 national Standard to refer to the application program that is using the interface.
347 [ISO/IEC 14392 {13}]
- 348 **2.2.2.89 user certificate [certificate]:** The public keys of a user, together with
349 some other information, rendered unforgeable by encipherment with the secret key
350 of the certification authority that issued it. [ISO/IEC 9594-8 {11}]
- 351 **2.2.2.90 value:** See *attribute value* in 2.2.2.15 and *OM attribute value* in 2.2.2.63.
- 352 **2.2.2.91 workspace:** A repository for instances of OM classes in the closures of
353 one or more packages associated with the workspace. [ISO/IEC 14360 {12}]

2.2.3 Abbreviations

For the purposes of this International Standard, the following abbreviations apply:

2.2.3.1 API: Application Program Interface.

2.2.3.2 ASN.1: Abstract Syntax Notation One.

2.2.3.3 AVA: Attribute Value Assertion.

2.2.3.4 BDCP: Basic Directory Contents Package.

2.2.3.5 BER: Basic Encoding Rules.

2.2.3.6 CA: Certification Authority.

2.2.3.7 CCITT: International Telegraph and Telephone Consultative Committee.

This organization has now been renamed as the Telecommunications Standardization Sector of the International Telecommunications Union (ITU-T).

2.2.3.8 DAP: Directory Access Protocol.

2.2.3.9 DIB: Directory Information Base.

2.2.3.10 DIT: Directory Information Tree.

2.2.3.11 DL: Delivery List.

2.2.3.12 DMD: Directory Management Domain.

2.2.3.13 DN: Distinguished Name.

2.2.3.14 DS: Directory Service.

2.2.3.15 DS API: Directory Services API (as defined in ISO/IEC 14392 {13} and ISO/IEC 14394 {B8}).

2.2.3.16 DS Package: Directory Service Package.

2.2.3.17 DSA: Directory System Agent.

2.2.3.18 DSP: Directory System Protocol.

2.2.3.19 DUA: Directory User Agent.

2.2.3.20 IA5: International Alphabet No. 5.

377 **2.2.3.21 ID:** Identifier.

378 **2.2.3.22 Interface:** The interface to directory services defined in ISO/IEC 14392
379 {13} and ISO/IEC 14394 {B8}.

380 **2.2.3.23 ISDN:** Integrated Services Digital Network.

381 **2.2.3.24 ISO:** International Organisation for Standardization.

382 **2.2.3.25 MDUP:** MHS Directory User Package.

383 **2.2.3.26 MHS:** message Handling System (as in ISO/IEC 10021 {B2}).

384 **2.2.3.27 MS:** Message Store.

385 **2.2.3.28 NBP:** Nonbasic Parameter.

386 **2.2.3.29 O/R:** Originator/Recipient.

387 **2.2.3.30 OM:** OSI abstract data Manipulation.

388 **2.2.3.31 OSI:** Open Systems Interconnection.

389 **2.2.3.32 POSIX.3:** IEEE Std 1003.3-1991 {14}.

390 **2.2.3.33 RDN:** Relative Distinguished Name.

391 **2.2.3.34 ROSE:** Remote Operations Service Element (see ISO/IEC 9072 and
392 CCITT Recommendation X.219: 1988 {3} and {4}).

393 **2.2.3.35 X.500 directory standards:** ISO/IEC 9594 and the 1988 CCITT X.500
394 Series Recommendations ({5}–{11}).

395 **2.2.3.36 SAP:** Strong Authentication Package.

Section 3: Test Assertions—General

3.1 Scope

There are no test assertions for this clause.

3.2 Normative References

There are no test assertions for this clause.

3.3 Conformance

3.3.1 Implementation Conformance

3.3.1.1 Requirements

There are no test assertions for this subclause.

3.3.1.2 Documentation

There are no test assertions for this subclause.

3.3.2 Conforming Implementation Options

GA01 For op in ds_abandon, ds_add_entry, ds_bind, ds_compare, ds_initialize, ds_list, ds_modify_entry, ds_modify_rdn, ds_receive_result, ds_read, ds_remove_entry, ds_search, ds_shutdown, ds_unbind, ds_version:

op exists in the implementation.

D01 The maximum number of outstanding asynchronous directory operations is documented.

3.3 Conformance

- 20 D02 Whether or not an asynchronous interface operation returns control
21 before the directory operation is submitted to the directory is docu-
22 mented.
- 23 D03 The manner in which the retrieval of completed directory operations
24 by ds_receive_result is prioritized is documented.
- 25 D04 The text and language of error messages is documented.
- 26 D05 Whether or not ds_abandon issues an Abandon directory operation
27 is documented.
- 28 D06 The OM classes permitted as values of the *Name* argument to inter-
face operations are documented.

29 **3.3.3 Application Conformance**

30 There are no test assertions for this section.

1 **Section 4: Test Assertions—Terminology and General Requirements**

2 There are no test assertions for this section.

Section 5: Test Assertions—Description

5.1 General

- GA02 For op in ds_abandon, ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_read, ds_remove_entry, ds_search:
The Abstract Service defined in the standards (with the same name as op) maps to op.
- GA03 For op in ds_bind, ds_unbind:
The Abstract Services DirectoryBind and DirectoryUnbind defined in the standards map to op, taking the values ds_bind and ds_unbind, respectively.

5.2 Initialization and Shutdown

- 001(A) When ds_initialize is invoked and completes successfully, then it returns a workspace that supports the standard Directory Service package.
- 002(B) When ds_initialize is invoked and completes successfully, then it returns a workspace that supports no packages other than the standard Directory Service package and no extensions that affect the behavior of applications that make use only of the standard features.
- GA04 For op in ds_compare, ds_list, ds_receive_result, ds_read, ds_search:
When the workspace returned by ds_initialize (and not subsequently extended) is used by op then any directory attribute values returned by op will be represented by OM objects of class **Encoding**.
- R01 The implementation allows for sections of code to invoke ds_version, ds_initialize, ds_shutdown independently, using distinct workspaces. [See 002(A) in 6.6.2.]
- 003(A) When ds_version is invoked, supplied with OSI Object Identifiers to identify additional packages or other extensions and those packages or extensions are supported by the implementation, then they are incorporated in the workspace.

5.3 Session

- 31 R01 When `ds_bind` is invoked, passing **Default-Session** as the Session
 32 argument then a session with default parameters will be started. [See
 33 003(A) in 6.4.3.]
- 34 001(B) When `ds_shutdown` is invoked and completes successfully, it will have
 35 the effect of deleting any private objects or service-generated public
 36 objects that are in the workspace.
- 37 002(A) The interface supports multiple concurrent sessions.

5.4 Context

- 38 GA05 For op in `ds_add_entry`, `ds_compare`, `ds_list`, `ds_modify_entry`, `ds_-`
 39 `modify_rdn`, `ds_read`, `ds_remove_entry`, `ds_search`:
 40 When op is invoked, then each of the `ServiceControls`
 41 defined in the Abstract Service maps onto an OM attribute in
 42 the **Context** that is used.
- 44 GA06 For op in `ds_add_entry`, `ds_compare`, `ds_list`, `ds_modify_entry`, `ds_-`
 45 `modify_rdn`, `ds_read`, `ds_remove_entry`, `ds_search`:
 46 When op is invoked, then the value of each component of the
 47 **Context** is determined and remains fixed throughout the direc-
 48 tory operation.
- 49 GA07 For op in `ds_add_entry`, `ds_compare`, `ds_list`, `ds_modify_entry`, `ds_-`
 50 `modify_rdn`, `ds_read`, `ds_remove_entry`, `ds_search`:
 51 When op is invoked, passing **Default-Context** as the *Context*
 52 argument, then it has the same effect as passing the **Context**
 53 object with default values.

5.5 Operation Parameters

- 54 GA08 For op in `ds_abandon`, `ds_add_entry`, `ds_compare`, `ds_list`, `ds_-`
 55 `modify_entry`, `ds_modify_rdn`, `ds_read`, `ds_remove_entry`, `ds_search`:
 56 The Abstract service defined in the standards (with the same
 57 name as op) has associated arguments that map to the
 58 corresponding arguments for op.
- 60 GA09 For op in `ds_bind`, `ds_unbind`:
 61 The Abstract Services `DirectoryBind` and `DirectoryUn-`
 62 `bind` defined in the standards has associated arguments that
 63 map to the corresponding arguments for op, taking the values
 64 `ds_bind` and `ds_unbind`, respectively.
- 65 GA10 For op in `ds_abandon`, `ds_add_entry`, `ds_bind`, `ds_compare`, `ds_list`,
 66 `ds_modify_entry`, `ds_modify_rdn`, `ds_read`, `ds_remove_entry`, `ds_-`
 67 `search`, `ds_unbind`:
 68 When op is specified to accept an instance of a particular OM

class as the value of an argument, then it will also accept an instance of any subclass of the OM class.

GA11 For op in ds_abandon, ds_add_entry, ds_bind, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_receive_result, ds_read, ds_remove_entry, ds_search, ds_unbind:

If the implementation recognizes a subclass of a class required in an argument (*new_class*) that is an extension:

When op(.. *new_class*..) is invoked, then *new_class* will be recognized.

GA12 For op in ds_abandon, ds_add_entry, ds_bind, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_receive_result, ds_read, ds_remove_entry, ds_search, ds_unbind:

If the implementation does not recognize a subclass of a class required in an argument (*new_class*) that is an extension:

When op(.. *new_class*..) is invoked, then it will ignore all OM attributes in *new_class* that are not permitted in the required OM class.

R01 The interface will always return private objects. (See GA21 in 5.6.)

5.5.1 Attribute and AVA

GA13 If the implementation supports the Basic Directory Contents Package:
For op in ds_compare, ds_list, ds_receive_result, ds_read, ds_search:

When the use of the BDCP has been negotiated and op is invoked and completes successfully, returning any attribute values that are defined in the BDCP, then they are represented as specified in the BDCP.

GA14 If the implementation supports the Strong Authentication Package:
For op in ds_compare, ds_list, ds_receive_result, ds_read, ds_search:

When the use of the SAP has been negotiated and op is invoked and completes successfully, returning any attribute values that are defined in the SAP, then they are represented as specified in the SAP.

GA15 For op in ds_compare, ds_list, ds_receive_result, ds_read, ds_search:
When op is invoked and completes successfully, returning any attribute values that are of ASN.1 simple type and are not known, then they are represented as shown in the table below.

	Type	Syntax
105	Bit String	String(Bit)
106	Boolean	Boolean
107	Integer	Integer
108	Null	Null
109	Object Identifier	String(Object Identifier)

- 110 GA16 For op in ds_compare, ds_list, ds_receive_result, ds_read, ds_search:
 111 When op is invoked and completes successfully, returning any
 112 attribute values whose attribute type is not known that is an
 113 ASN.1 structured type, then they are represented by OM
 114 objects of class **Encoding**.
- 115 R01 The OM class **AVA** is a subclass of OM class **Attribute** constrained
 116 to have exactly one value. The OM attributes of the instance of **AVA**
 117 that is returned are as stated in R01 in 7.7. (See GA40 in 7.7.)
- 118 R02 The OM object of OM class **Attribute-List** contains a single, multi-
 119 valued OM attribute whose values are OM objects of OM class **Attri-**
 120 **bute**. (See R01 in 7.6.)

121 5.5.2 Entry Information Selection

- 122 GA17 For op in ds_read, ds_search:
 123 When op is invoked and completes successfully, then attribute
 124 types will always be returned.
- 125 GA18 For op in ds_read, ds_search:
 126 When op(.. *selection*..) is invoked and completes successfully
 127 and the constant **Select-All-Types** is passed instead of *selec-*
 128 *tion* then just the types of all attributes will be returned.
- 129 GA19 For op in ds_read, ds_search:
 130 When op(.. *selection*..) is invoked and completes successfully
 131 and the constant **Select-All-Types-And-Values** is passed
 132 instead of *selection* then the types and values of all attributes
 133 will be returned.

134 5.5.3 Name

- 135 R01 For op in ds_compare, ds_list, ds_read, ds_search:
 136 When op(.. *context*.. *name*..) is invoked and **Dont-**
 137 **Dereference-Aliases** in *context* is not set, then any aliases in
 138 *name* will be dereferenced. (See GA42 in 7.10.)
- 139 R02 For op in ds_compare, ds_list, ds_read, ds_search:
 140 When op(.. *context*.. *name*..) is invoked and **Dont-**
 141 **Dereference-Aliases** in *context* is set, then any aliases in
 142 *name* will not be dereferenced. (See GA42 in 7.10.)
- 143 GA20 For op in ds_add_entry, ds_modify_entry, ds_modify_rdn, ds_
 144 remove_entry:

145 When op is invoked, then any aliases in the name will not be
dereferenced.

146 5.6 Operation Results

147 R01 For op in ds_abandon, ds_add_entry, ds_bind, ds_compare, ds_list,
148 ds_modify_entry, ds_modify_rdn, ds_receive_result, ds_read, ds_
149 remove_entry, ds_search, ds_unbind, ds_version:

150 When op is invoked and completes successfully, then a status
151 result is returned. (See GA24 in 5.6.3.)

152 GA21 For op in ds_bind, ds_compare, ds_list, ds_receive_result, ds_read,
153 ds_search:

154 When op is invoked and the associated directory operation
155 completes successfully, then op will return private objects in
156 the workspace associated with the directory session in the invo-
157 cation of ds_bind.

158 5.6.1 The Invoke ID Parameter

159 GA22 If the implementation supports asynchronous operation:

160 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry,
161 ds_modify_rdn, ds_receive_result, ds_read, ds_remove_entry,
162 ds_search:

163 When op is invoked and asynchronous operation is
164 selected and the associated directory operation completes
165 successfully, then the numerical value returned is
166 guaranteed to be unique amongst all outstanding direc-
167 tory operations in a given session. The value is such as
168 could be returned from the Remote Operations Service
169 Element (ROSE) defined in ISO/IEC 9072 ({3} and {4}).

170 5.6.2 Result

171 GA23 For op in ds_compare, ds_list, ds_receive_result, ds_read, ds_search:
172 When op is invoked and the associated directory operation does
173 not complete successfully, then a result is not returned.

174 R01 For op in ds_compare, ds_list, ds_receive, ds_read, ds_search:
175 When op is invoked and the associated directory operation does
176 not complete successfully, then the errors returned are reported
177 in the status result. (See, GA24 in 5.6.3.)

178 R02 If the implementation supports asynchronous operation:

179 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry,
180 ds_modify_rdn, ds_search, ds_read, ds_remove_entry:

181 When op is invoked and asynchronous operation is
182 selected, then the result of the associated directory

operation is returned by a later invocation of `ds_receive_result`. (See GA28 in 5.7.)

5.6.3 Status

GA24 For `op` in `ds_abandon`, `ds_add_entry`, `ds_bind`, `ds_compare`, `ds_initialize`, `ds_list`, `ds_modify_entry`, `ds_modify_rdn`, `ds_receive_result`, `ds_read`, `ds_remove_entry`, `ds_search`, `ds_shutdown`, `ds_unbind`, `ds_version`:

When `op` is invoked, then the status result that is returned is one of the following: the constant **success**, the constant **no_workspace**, the constant **invalid_workspace**, an attribute error represented by a private object of class **Attribute-Error**, a referral represented by a private object of class **Referral**, or an error represented by a private object of one of the subclasses of class **Error**.

5.7 Synchronous and Asynchronous Operations

R01 If the implementation supports asynchronous operation:
For `op` in `ds_add_entry`, `ds_compare`, `ds_list`, `ds_modify_entry`, `ds_modify_rdn`, `ds_search`, `ds_read`, `ds_remove_entry`:

When the value of the **Asynchronous** OM attribute in the *Context* passed to `op` is **true** then the directory operation is asynchronous. The OM attributes of the instance of **Context** that is returned are as stated in R01 in 7.10. (See GA56 in 7.10.)

R02 If the implementation supports asynchronous operation:
For `op` in `ds_add_entry`, `ds_compare`, `ds_list`, `ds_modify_entry`, `ds_modify_rdn`, `ds_search`, `ds_read`, `ds_remove_entry`:

When the value of the **Asynchronous** OM attribute in the *Context* passed to `op` is **false** then the directory operation is synchronous. The OM attributes of the instance of **Context** that is returned are as stated in R01 in 7.10. (See GA55 in 7.10.)

GA25 If the implementation does not support asynchronous operation:
For `op` in `ds_add_entry`, `ds_compare`, `ds_list`, `ds_modify_entry`, `ds_modify_rdn`, `ds_search`, `ds_read`, `ds_remove_entry`:
The directory operation associated with `op` is synchronous.

GA26 If the implementation supports asynchronous operation:
For `op` in `ds_add_entry`, `ds_compare`, `ds_list`, `ds_modify_entry`, `ds_modify_rdn`, `ds_search`, `ds_read`, `ds_remove_entry`:
When the value of the **Asynchronous** OM attribute is not specified in the *Context* passed to `op` then `op`

- behaves as if the value of the OM attribute was **false** and the directory operation is synchronous.
- GA27 If the implementation supports asynchronous operation:
 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_search, ds_read, ds_remove_entry:
 When op is invoked and asynchronous operation is selected and op completes successfully, then the *invoke_id* of the directory operation is returned by op.
- GA28 If the implementation supports asynchronous operation:
 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_search, ds_read, ds_remove_entry:
 When op is invoked and asynchronous operation is selected and completes successfully, then the *invoke_id* returned uniquely identifies the pending result and can be passed to ds_abandon to cause the directory operation to be abandoned, or passed to ds_receive_result to receive the results for that directory operation.
- GA29 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_search, ds_read, ds_remove_entry:
 If the implementation defines that all interface operations return control after an asynchronous directory operation has been submitted to the underlying directory service:
 When op is invoked and asynchronous operation is selected, then it returns control after waiting for the associated directory operation to be submitted to the underlying directory service.
 Otherwise:
 When op is invoked and asynchronous operation is selected, then it returns control as soon as possible.
- D07 The limit on the number of asynchronous directory operations that may be outstanding at any one time in any one session is documented.
- 001(C) If the implementation supports asynchronous operation:
 The constant **Max-Outstanding-Operations** is defined as a value of at least one.
- 002(C) If the implementation does not support asynchronous operation:
 The constant **Max-Outstanding-Operations** is defined with the value zero.
- R03 The value of **Max-Outstanding-Operations** indicates the presence or absence of asynchronous operation. [See both 001(C) in 5.7 and 002(C) in 5.7.]
- GA30 If the implementation supports asynchronous operation:
 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_search, ds_read, ds_remove_entry:
 When op is invoked and asynchronous operation is selected and the number of outstanding directory operations is equal to **Max-Outstanding-Operations** then it

returns an instance of class **Library-Error** with **too-many-operations** value of the **Problem** attribute.

- GA31 If the implementation supports asynchronous operation:
 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_search, ds_read, ds_remove_entry:
 When op is invoked and asynchronous operation is selected and an error is detected before the associated asynchronous request is submitted to the directory, then op returns control immediately and no outstanding directory operation is generated.
- GA32 If the implementation supports asynchronous operation:
 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_search, ds_read, ds_remove_entry:
 When op is invoked and asynchronous operation is selected and an error occurs after the associated asynchronous request is submitted to the directory, then errors are reported by ds_receive_result.

5.8 Automatic Connection Management

There are no test assertions for this clause.

5.9 Automatic Continuation and Referral Handling

- R01 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_read, ds_remove_entry, ds_search:
 When continuation references are received during the execution of op then the implementation will use them to attempt to contact the other DSA(s) whenever practical. (See GA58 in 7.10.)
- GA33 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_read, ds_remove_entry, ds_search:
 When continuation references have all been used by op to attempt to contact other DSA(s), then op returns a result or error.
- R02 For op in ds_add_entry, ds_compare, ds_list, ds_modify_entry, ds_modify_rdn, ds_read, ds_remove_entry, ds_search:
 When **Automatic-Continuations** is **false** in the **Context** object and op is invoked, then op returns all the continuation references. (See GA57 in 7.10.)

Section 6: Test Assertions—Interface Operations

NOTE: There are a number of instantiations in this section that come from general assertions in Section 5 that are conditionally extended assertions. These assertions can only be reliably tested by examining the protocol going between the DUA and the DSA (DAP) or between the DSAs (DSP) which is not a required feature on all implementations. On systems that support the X.500 protocols, these assertions can be tested portably. The assertions are therefore classified as being base assertions for this class of systems and extended assertions for other systems. This applies to all the assertions relating to the **Context** object that are followed by a testing requirements statement explaining how to test the assertion by examining the protocol.

There are other assertions that cannot be tested portably because they require the implementation to force the production of a referral (i.e., assertions that involve the generation of **Access-Point**, **Presentation-Address**, **Continuation-Reference**, **Operation-Progress**, **Partial-Outcome-Qualifier**). Such assertions are marked as being extended, unconditionally.

6.1 Datatype Definitions

There are no test assertions for this clause.

6.2 Abandon

6.2.1 Synopsis

001(A) `ds_abandon` exists in the implementation. (See GA01 in 3.3.2.)

002({X.500_PROTOCOLS}?A:B) The Abstract Service Abandon defined in the standards maps to `ds_abandon`. (See GA02 in 5.1.)

6.2.2 Description

001(C) If the implementation supports asynchronous operation:
When `ds_abandon(, invoke_id)` is invoked and completes successfully, then the result of the outstanding asynchronous directory operation identified by `invoke_id` is abandoned.

002(D) If the implementation supports asynchronous operation:
When `ds_abandon(, invoke_id)` is invoked and completes successfully, then the result of the outstanding asynchronous

directory operation identified by *invoke_id* is never returned by *ds_receive_result*.

6.2.3 Arguments In

- 001(D) If the implementation recognizes a subclass of a class required in an argument (*new_class*) that is an extension:
When *ds_abandon*(.. *new_class*..) is invoked, then *new_class* will be recognized. (See GA11 in 5.5.)
- 002(C) If the implementation does not recognize a subclass of a class required in an argument (*new_class*) that is an extension:
When *ds_abandon*(.. *new_class*..) is invoked, then it will ignore all OM attributes in *new_class* that are not permitted in the required OM class. (See GA12 in 5.5.)
- 003({X.500_PROTOCOLS}?A:B) The Abstract Service Abandon defined in the standards has associated arguments that map to the corresponding arguments for *ds_abandon*. (See GA08 in 5.5.)
- 004(A) When *ds_abandon* is specified to accept an instance of a particular OM class as the value of an argument, then it will also accept an instance of any subclass of the OM class. (See GA10 in 5.5.)

6.2.4 Results

- 001(A) When *ds_abandon* is invoked, then the status result that is returned is one of the following: the constant **success**, the constant **no_workspace**, the constant **invalid_workspace**, an attribute error represented by a private object of class **Attribute-Error**, a referral represented by a private object of class **Referral**, or an error represented by a private object of one of the subclasses of class **Error**. (See GA24 in 5.6.3.)
- 002(A) When *ds_abandon* is invoked and completes successfully, then the returned value of type *ds_result_type* will be **success**. (See GA94 in 8.1.)
- 003(A) When *ds_abandon* is invoked and does not complete successfully and has been given a *session* argument containing a valid workspace, then it returns a private object of class **Error**, **Attribute-Error**, or **Referral** in that workspace. (See GA95 in 8.1.)

6.2.5 Errors

- 001(A) When *ds_abandon* is invoked with a *session* argument that does not contain a valid workspace, then **no_workspace** is returned. (See GA97 in 8.1.)
- 002(C) If the implementation supports asynchronous operation:
When *ds_abandon* is invoked and an attempt was made to abandon the directory operation *ds_add_entry* for which this is

prohibited, then an **Abandon-Failed** error is returned with the value of the attribute **Problem** set to **cannot-abandon**. The OM attributes of the instance of **Abandon-Failed** that is returned are as stated in R01 in 8.4. (See GA102 in 8.4.)

- 003(C) If the implementation supports asynchronous operation:
When `ds_abandon` is invoked and an attempt was made to abandon directory operation `ds_compare` that could not be abandoned, then an **Abandon-Failed** error is returned with the value of the **Problem** attribute set to **cannot-abandon**. The OM attributes of the instance of **Abandon-Failed** that is returned are as stated in R01 in 8.4. (See GA103 in 8.4.)
- 004(C) If the implementation supports asynchronous operation:
When `ds_abandon` is invoked and an attempt was made to abandon directory operation `ds_list` that could not be abandoned, then an **Abandon-Failed** error is returned with the value of the **Problem** attribute set to **cannot-abandon**. The OM attributes of the instance of **Abandon-Failed** that is returned are as stated in R01 in 8.4. (See GA103 in 8.4.)
- 005(C) If the implementation supports asynchronous operation:
When `ds_abandon` is invoked and an attempt was made to abandon the directory operation `ds_modify_entry` for which this is prohibited, then an **Abandon-Failed** error is returned with the value of the attribute **Problem** set to **cannot-abandon**. The OM attributes of the instance of **Abandon-Failed** that is returned are as stated in R01 in 8.4. (See GA102 in 8.4.)
- 006(C) If the implementation supports asynchronous operation:
When `ds_abandon` is invoked and an attempt was made to abandon the directory operation `ds_modify_rdn` for which this is prohibited, then an **Abandon-Failed** error is returned with the value of the attribute **Problem** set to **cannot-abandon**. The OM attributes of the instance of **Abandon-Failed** that is returned are as stated in R01 in 8.4. (See GA102 in 8.4.)
- 007(C) If the implementation supports asynchronous operation:
When `ds_abandon` is invoked and an attempt was made to abandon directory operation `ds_read` that could not be abandoned, then an **Abandon-Failed** error is returned with the value of the **Problem** attribute set to **cannot-abandon**. The OM attributes of the instance of **Abandon-Failed** that is returned are as stated in R01 in 8.4. (See GA103 in 8.4.)
- 008(C) If the implementation supports asynchronous operation:
When `ds_abandon` is invoked and an attempt was made to abandon the directory operation `ds_remove_entry` for which this is prohibited, then an **Abandon-Failed** error is returned with the value of the attribute **Problem** set to **cannot-abandon**. The OM attributes of the instance of **Abandon-Failed** that is returned are as stated in R01 in 8.4. (See GA102 in 8.4.)

- 009(C) If the implementation supports asynchronous operation:
 113 When `ds_abandon` is invoked and an attempt was made to
 114 abandon directory operation `ds_search` that could not be aban-
 115 doned, then an **Abandon-Failed** error is returned with the
 116 value of the **Problem** attribute set to **cannot-abandon**. The
 117 OM attributes of the instance of **Abandon-Failed** that is
 118 returned are as stated in R01 in 8.4. (See GA103 in 8.4.)
- 010([X.500_PROTOCOLS]?A:B) When `ds_abandon` is invoked and a communi-
 119 cations error occurs (including those arising in Remote Operation,
 120 Association Control, Presentation, Session, and Transport), then a
 121 **Communications-Error** is returned with the value of the **Problem**
 122 attribute set to **communications-problem**. The OM attributes of
 123 the instance of **Communications-Error** that is returned are as
 124 stated in R01 in 8.7. (See GA119 in 8.7.)
- 011(A) When `ds_abandon(session,)` is invoked and an invalid *session* was
 126 supplied, then a **Library-Error** is returned with the value of the
 127 **Problem** attribute set to **bad-session**. The OM attributes of the
 128 instance of **Library-Error** that is returned are as stated in R01 in
 129 8.8. (See GA125 in 8.8.)
- 012(B) When `ds_abandon` is invoked and the interface cannot clear a tran-
 131 sient system error by retrying the affected system call, then a
 132 **Library-Error** is returned with the value of the **Problem** attribute
 133 set to **miscellaneous**. The OM attributes of the instance of **Library-**
 134 **Error** that is returned are as stated in R01 in 8.8. (See GA127 in
 135 8.8.)
- 013(C) If the implementation supports asynchronous operation:
 137 When `ds_abandon` is invoked and an attempt to start a syn-
 138 chronous directory operation was made while there were out-
 139 standing asynchronous directory operations, then a **Library-**
 140 **Error** is returned with the value of the **Problem** attribute set
 141 to **mixed-synchronous**. The OM attributes of the instance of
 142 **Library-Error** that is returned are as stated in R01 in 8.8.
 143 (See GA129 in 8.8.)
- 014(A) When `ds_abandon` is invoked and an attempt was made to use
 145 optional functionality that is not available in the implementation,
 146 then a **Library-Error** is returned with the value of the **Problem**
 147 attribute set to **not-supported**. The OM attributes of the instance of
 148 **Library-Error** that is returned are as stated in R01 in 8.8. (See
 149 GA130 in 8.8.)
- 015(B) When a system error occurs and `ds_abandon` generates an instance of
 151 OM class **System-Error**, then its **Problem** attribute will identify the
 152 cause of the failure. The OM attributes of the instance of **System-**
 153 **Error** that is returned are as stated in R01 in 8.13. (See GA156 in
 154 8.13.)
- 016(B) When `ds_abandon` is invoked and a transient error occurs, then the
 156 affected directory operation will be retried and these errors will not be
 157 returned. (See GA157 in 8.13.)

6.2.6 Cross-References

159 There are no test assertions for this subclause.

160 6.3 Add-Entry

161 6.3.1 Synopsis

162 001(A) `ds_add_entry` exists in the implementation. (See GA01 in 3.3.2.)

163 6.3.2 Description

164 001(A) When `ds_add_entry(,, entry,)` is invoked, where `entry` is an object that
165 conforms to the directory schema, and the associated directory operation
166 completes successfully, then a leaf entry is added to the directory.

167 002(A) When `ds_add_entry(,, entry,)` is invoked, where `entry` is an alias that
168 conforms to the directory schema, and the associated directory operation
169 completes successfully, then a leaf entry is added to the directory.

170 003({X.500_PROTOCOLS}?A:B) The Abstract Service `AddEntry` defined in the
171 standards maps to `ds_add_entry`. (See GA02 in 5.1.)

172 004(C) If the implementation does not support asynchronous operation:
173 The directory operation associated with `ds_add_entry` is syn-
174 chronous. (See GA25 in 5.7.)

175 005(D) If the implementation defines that all interface operations return con-
176 trol after an asynchronous directory operation has been submitted to
177 the underlying directory service:

178 When `ds_add_entry` is invoked and asynchronous operation is
179 selected, then it returns control after waiting for the associated
180 directory operation to be submitted to the underlying directory
181 service.

182 Otherwise:

183 When `ds_add_entry` is invoked and asynchronous operation is
184 selected, then it returns control as soon as possible. (See GA29
185 in 5.7.)

186 006(C) If the implementation supports asynchronous operation:

187 When `ds_add_entry` is invoked and asynchronous operation is
188 selected and an error is detected before the associated asyn-
189 chronous request is submitted to the directory, then `ds_add_`
190 `entry` returns control immediately and no outstanding directory
191 operation is generated. (See GA31 in 5.7.)

192 007(C) If the implementation supports asynchronous operation:

193 When `ds_add_entry` is invoked and asynchronous operation is
194 selected and an error occurs after the associated asynchronous
195 request is submitted to the directory, then errors are reported
196 by `ds_receive_result`. (See GA32 in 5.7.)

- 008(A) When continuation references have all been used by `ds_add_entry` to attempt to contact other DSAs, then `ds_add_entry` returns a result or error. (See GA33 in 5.9.)

6.3.3 Arguments In

- 001(D) If the implementation recognizes a subclass of a class required in an argument (*new_class*) that is an extension:
When `ds_add_entry(.. new_class..)` is invoked, then *new_class* will be recognized. (See GA11 in 5.5.)
- 002(C) If the implementation does not recognize a subclass of a class required in an argument (*new_class*) that is an extension:
When `ds_add_entry(.. new_class..)` is invoked, then it will ignore all OM attributes in *new_class* that are not permitted in the required OM class. (See GA12 in 5.5.)
- 003(B) When `ds_add_entry` is invoked, then the `ServiceControls` defined in the Abstract Service each map onto an OM attribute in the **Context** that is used. (See GA05 in 5.4.)
- 004(A) When `ds_add_entry` is invoked, then the value of each component of the **Context** is determined and remains fixed throughout the directory operation. (See GA06 in 5.4.)
- 005(A) When `ds_add_entry` is invoked, passing **Default-Context** as the *Context* argument, then it has the same effect as passing the **Context** object with default values. (See GA07 in 5.4.)
- 006({X.500_PROTOCOLS}?A:B) The Abstract Service `AddEntry` defined in the standards has associated arguments that map to the corresponding arguments for `ds_add_entry`. (See GA08 in 5.5.)
- 007(A) When `ds_add_entry` is specified to accept an instance of a particular OM class as the value of an argument, then it will also accept an instance of any subclass of the OM class. (See GA10 in 5.5.)
- 008(A) When `ds_add_entry` is invoked, then any aliases in the name will not be dereferenced. (See GA20 in 5.5.3.)
- 009(C) If the implementation supports asynchronous operation:
When the value of the **Asynchronous** OM attribute is not specified in the *Context* passed to `ds_add_entry`, then `ds_add_entry` behaves as if the value of the OM attribute was **false** and the directory operation is synchronous. (See GA26 in 5.7.)
- 010({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and **Chaining-Prohibited** is set, then chaining, and other methods of distributing the request around the directory are prohibited. (See GA41 in 7.10.)
- Testing Requirements: In a distributed system this assertion can be tested by checking the Directory Access Protocol or the Directory System Protocol.

011({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and
Dont-Use-Copy is set, then the request is to be satisfied only by
 access to directory entries and not copies maintained in other DSAs or
 locally cached copies. (See GA43 in 7.10.)

Testing Requirements: In a distributed system this assertion can be
 tested by checking the Directory Access Protocol or the Directory Sys-
 tem Protocol.

012({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and
Local-Scope is set, then the request is to be satisfied locally, the
 meaning of which is configured by an administrator. (See GA44 in
 7.10.)

Testing Requirements: In a distributed system this assertion can be
 tested by checking the Directory Access Protocol or the Directory Sys-
 tem Protocol.

013({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and
Prefer-Chaining is set, then chaining is preferred to referrals when
 necessary. (See GA45 in 7.10.)

Testing Requirements: In a distributed system this assertion can be
 tested by checking the Directory Access Protocol or the Directory Sys-
 tem Protocol.

014({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and
Priority is set to **low**, then the implementation will attempt to
 satisfy the request at a low priority relative to the other requests.
 (See GA46 in 7.10.)

Testing Requirements: In a distributed system this assertion can be
 tested by checking the Directory Access Protocol or the Directory Sys-
 tem Protocol.

015({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and
Priority is set to **medium**, then the implementation will attempt to
 satisfy the request at a medium priority relative to the other requests.
 (See GA47 in 7.10.)

Testing Requirements: In a distributed system this assertion can be
 tested by checking the Directory Access Protocol or the Directory Sys-
 tem Protocol.

016({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and
Priority is set to **high**, then the implementation will attempt to
 satisfy the request at a high priority relative to the other requests.
 (See GA48 in 7.10.)

Testing Requirements: In a distributed system this assertion can be
 tested by checking the Directory Access Protocol or the Directory Sys-
 tem Protocol.

017({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and **Scope-Of-Referral** is set to **country**, then referrals will be limited to DSAs in the originating country. (See GA49 in 7.10.)

Testing Requirements: In a distributed system this assertion can be tested by checking the Directory Access Protocol or the Directory System Protocol.

018({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and **Scope-Of-Referral** is set to **DMD**, then referrals will be limited to DSAs within the originating DMD. (See GA50 in 7.10.)

Testing Requirements: In a distributed system this assertion can be tested by checking the Directory Access Protocol or the Directory System Protocol.

019({X.500_PROTOCOLS}?A:B) When `ds_add_entry(, context,,)` is invoked and **Scope-Of-Referral** is not set, then referrals will not be limited. (See GA51 in 7.10.)

Testing Requirements: In a distributed system this assertion can be tested by checking the Directory Access Protocol or the Directory System Protocol.

020({X.500_PROTOCOLS}?A:B) When the time specified in **Time-Limit** has elapsed after `ds_add_entry(, context,,)` has been invoked, then the associated directory operation will return a **Service-Error (time-limit-exceeded)**. (See GA53 in 7.10.)

021(A) When `ds_add_entry(, context,,)` is invoked and **Asynchronous** is set to **false**, then the associated directory operation will be performed sequentially, the application being blocked until a result or error is returned. (See GA55 in 7.10.)

022(C) If the implementation supports asynchronous operation:
When `ds_add_entry(, context,,)` is invoked and **Asynchronous** is set to **true**, then the associated directory operation will be performed asynchronously (**nonblocking**). (See GA56 in 7.10.)

023(B) When `ds_add_entry(, context,,)` is invoked and **Automatic-Continuation** is **false**, then the associated directory operation will return all continuation references to the application. (See GA57 in 7.10.)

024(B) When `ds_add_entry(, context,,)` is invoked and **Automatic-Continuation** is **true**, then the associated directory operation will process continuation references automatically and return the subsequent results whenever practical. (See GA58 in 7.10.)

6.3.4 Arguments Out

001(C) If the implementation supports asynchronous operation:
When `ds_add_entry` is invoked and the associated directory operation completes successfully and asynchronous operation is

selected, then the numerical value returned is guaranteed to be unique among all outstanding directory operations in a given session. The value is such as could be returned from the Remote Operations Service Element (ROSE) defined in ISO/IEC 9072 ({3} and {4}). (See GA22 in 5.6.1.)

002(C) If the implementation supports asynchronous operation:
When `ds_add_entry` is invoked and `ds_add_entry` completes successfully and asynchronous operation is selected, then the `invoke_id` of the directory operation is returned by `ds_add_entry`. (See GA27 in 5.7.)

003(C) If the implementation supports asynchronous operation:
When `ds_add_entry` is invoked and asynchronous operation is selected and completes successfully, then the `invoke_id` returned uniquely identifies the pending result and can be passed to `ds_abandon` to cause the directory operation to be abandoned, or passed to `ds_receive_result` to receive the results for that directory operation. (See GA28 in 5.7.)

6.3.5 Results

001(A) When `ds_add_entry` is invoked, then the status result that is returned is one of the following: the constant **success**, the constant **no_workspace**, the constant **invalid_workspace**, an attribute error represented by a private object of class **Attribute-Error**, a referral represented by a private object of class **Referral**, or an error represented by a private object of one of the subclasses of class **Error**. (See GA24 in 5.6.3.)

002(A) When `ds_add_entry` is invoked and completes successfully, then the returned value of type `ds_result_type` will be **success**. (See GA94 in 8.1.)

003(A) When `ds_add_entry` is invoked and does not complete successfully and has been given a *session* argument containing a valid workspace, then it returns a private object of class **Error**, **Attribute-Error**, or **Referral** in that workspace. (See GA95 in 8.1.)

6.3.6 Errors

001(A) When `ds_add_entry` is invoked with a *session* argument that does not contain a valid workspace, then **no_workspace** is returned. (See GA97 in 8.1.)

002(C) If the implementation supports asynchronous operation:
When `ds_add_entry` is invoked and asynchronous operation is selected and the number of outstanding directory operations is equal to **Max-Outstanding-Operations**, then it returns an instance of class **Library-Error** with **too-many-operations** value of the **Problem** attribute. (See GA30 in 5.7.)

- 003(D) If automatic connection management is supported:
 When `ds_add_entry` is invoked, then the associated directory operation will return the same errors as `ds_bind`. (See GA99 in 8.1.)
- 004(A) When `ds_add_entry` is invoked and an automatic referral list is followed, then the associated directory operation will return the same errors as `ds_bind`. (See GA100 in 8.1.)
- 005(A) When `ds_add_entry` generates an instance of **Attribute-Error**, then it will contain the name of the directory entry to which the directory operation was being applied when the failure occurred in the **Object-Name** attribute. The OM attributes of the instance of **Attribute-Error** that is returned are as stated in R01 in 8.5. (See GA104 in 8.5.)
- 006(A) When `ds_add_entry` generates an instance of **Attribute-Error**, then it will document the attribute-related problems encountered in the **Problems** attribute. The OM attributes of the instance of **Attribute-Error** that is returned are as stated in R01 in 8.5. (See GA105 in 8.5.)
- 007(A) When `ds_add_entry` generates an instance of **Attribute-Error**, then several problems at once can be reported, all related to the **Object-Name** attribute of this class. The OM attributes of the instance of **Attribute-Error** that is returned are as stated in R01 in 8.5. (See GA106 in 8.5.)
- 008(A) When `ds_add_entry` generates an instance of **Attribute-Problem**, then it will identify the type of attribute with which the problem is associated in the **Attribute-Type** attribute. The OM attributes of the instance of **Attribute-Problem** that is returned are as stated in R01 in 8.6. (See GA107 in 8.6.)
- 009(A) When `ds_add_entry` generates an instance of **Attribute-Problem**, then it will contain the attribute value with which the problem is associated, in the **Attribute-Value** attribute. The OM attributes of the instance of **Attribute-Problem** that is returned are as stated in R01 in 8.6. (See GA108 in 8.6.)
- 010(A) When `ds_add_entry` is invoked and an attempt was made to add an attribute or value that is already present in the directory entry in question, then an **Attribute-Problem** error is returned with the value of the **Problem** attribute set to **attribute-or-value-already-exists**. The OM attributes of the instance of **Attribute-Problem** that is returned are as stated in R01 in 8.6. (See GA109 in 8.6.)
- 011(A) When `ds_add_entry` is invoked and the attribute value does not conform to the constraints imposed by the standards (see ISO/IEC 9594-1 to ISO/IEC 9594-8 {5}–{11}) (e.g., the value exceeds its upper bound), then an **Attribute-Problem** error is returned with the value of the **Problem** attribute set to **constraint-violation**. The OM attributes of the instance of **Attribute-Problem** that is returned are as stated in R01 in 8.6. (See GA110 in 8.6.)