

IEEE Standard for Information Technology— Test Methods for Measuring Conformance to Directory Services C Language Interfaces— Binding for Application Program Interface (API)

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: Test methods for the C language binding defined in IEEE Std 1327.2-1993 for the applications interface to directory services defined in IEEE Std 1224.2-1993 are specified.

Keywords: API, application portability, C language, open systems, directory, portable application, test methods

The 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 1-55937-316-4

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 the 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, the 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 are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Contents

	PAGE
Introduction	v
Section 1: General	1
1.1 Scope	1
1.2 Normative References	1
1.3 Conformance	2
Section 2: Terminology and General Requirements	5
2.1 Conventions	5
2.2 Definitions	11
Section 3: Test Assertions—General	23
Section 4: Test Assertions—Terminology and General Requirements	25
4.1 Conventions	25
4.2 Definitions	26
Section 5: Test Assertions—The Generic Interface	27
5.1 Introduction	27
5.2 Interface Functions	27
5.2.1 Overview	27
5.2.2 Common Datatypes	27
5.2.3 Abandon	27
5.2.4 Add Entry	28
5.2.5 Bind	29
5.2.6 Compare	29
5.2.7 Initialize	30
5.2.8 List	31
5.2.9 Modify Entry	31
5.2.10 Modify RDN	32
5.2.11 Read	33
5.2.12 Receive Result	33
5.2.13 Remove Entry	34
5.2.14 Search	35
5.2.15 Shutdown	36
5.2.16 Unbind	36
5.2.17 Version	37
5.3 Directory Service Package	38
5.3.1 Object Identifiers	38
5.3.2 Enumeration Types and Constants	39
5.3.3 OM Object Constants	41

5.3.4	Integer Constants	42
5.3.5	Class Constraints	42
5.4	The <xds.h> Header	43
Section 6: Test Assertions—The X.500 Directory Service		45
6.1	Overview	45
6.2	Basic Directory Contents Package	45
6.2.1	Directory Object and Attribute Object Identifiers	45
6.2.2	OM Object and Attribute Type Identifiers	46
6.2.3	Directory Enumeration Types and Constants	47
6.2.4	Integer Constants	47
6.3	Strong Authentication Package	48
6.3.1	Directory Object and Attribute Object Identifiers	48
6.3.2	OM Object and Attribute Type Identifiers	49
6.3.3	Enumeration Types and Constants	50
6.3.4	Integer Constants	50
6.4	MHS Directory User Package	50
6.4.1	Directory Object and Attribute Object Identifiers	50
6.4.2	OM Object and Attribute Type Identifiers	51
6.4.3	Enumeration Types and Constants	51
6.5	Header Files	52
6.5.1	The <xdsbdc.h> Header	52
6.5.2	The <xdssap.h> Header	52
6.5.3	The <xdsmdup.h> header	52
Section 7: Test Assertions—Definitions of Constants		55
7.1	Introduction	55
7.2	Directory Service Package	55
7.2.1	Object and Attribute Type Identifiers	55
7.2.2	Enumeration Types and Constants	55
7.2.3	OM Object Constants	58
7.2.4	Integer Constants	59
7.3	Basic Directory Contents Package	60
7.4	Strong Authentication Package	60
7.5	MHS Directory User Package	60
Annex A (informative) Bibliography		61
Alphabetic Topical Index		63

TABLES

Table 2-1 – C Naming Conventions 8

Introduction

(This introduction is not a normative part of IEEE Std 1328.2-1993, IEEE Standard for Information Technology—Test Methods for Measuring Conformance to Directory Services C Language Interfaces—Binding for Application Program Interface (API), but is included for information only.)

Purpose

The purpose of this standard is to define test methods for the C language binding contained in IEEE Std 1327.2-1993 {8} for the language-independent specification contained in IEEE Std 1224.2-1993 {4} of the application programming interface (API) to directory services.

A directory is a distributed collection of information, that programs can access in order to make queries or updates. IEEE Std 1224.2-1993 {4} defines, in programming language independent terms, an API to directory services. This API is known as the Directory Services API (DS API). IEEE Std 1327.2-1993 {8} defines a C language binding for the 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 1988 CCITT X.500 Series recommendations and ISO/IEC 9594: 1990. IEEE Std 1224.2-1993 {4} prescribes how the DS API is to be used to access the particular directory service defined in ISO/IEC 9594: 1990 and indicates how it may be used to access other directory services that conform to the same abstract model. Nothing in IEEE Std 1224.2-1993 {4} or 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.

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.

Related Standards

IEEE Std 1224.2-1993 {4} 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 IEEE Std 1327.2-1993 {8}. This standard applies to test methods for measuring conformance to that programming language binding specification.

1 IEEE Std 1326.2-1993 {6} specifies test methods for the language-independent
2 specification contained in IEEE Std 1224.2-1993 {4}. A set of test methods for the
3 C binding to IEEE Std 1224.2-1993 {4} must satisfy the requirements of IEEE Std
4 1326.2-1993 {6}, as well as conforming to this standard.

5 The API defined in IEEE Std 1224.2-1993 {4} uses the mechanism for OSI abstract
6 data Manipulation (OM) defined in IEEE Std 1224-1993 {B13}. IEEE Std
7 1326-1993 {5} defines the requirements that apply to test methods for measuring
8 conformance to IEEE Std 1224-1993 {B13}. A C language binding to IEEE Std
9 1224-1993 {B13} is defined in IEEE Std 1327-1993 {7}. IEEE Std 1328-1993 {9}
10 defines the requirements that apply to test methods for measuring conformance to
11 IEEE Std 1327-1993 {7}.

12 IEEE Std 1003.3-1991 {3} defines the general requirements that shall apply to test
13 methods for measuring conformance to POSIX. A set of test methods used to mea-
14 sure conformance to IEEE Std 1224.2-1993 {4} must satisfy the requirements of
15 IEEE Standard 1003.3, as well as conforming to this standard.

16 **Overview**

17 For each section of IEEE Std 1327.2-1993 {8}, this standard contains a correspond-
18 ing section containing test assertions, in accordance with IEEE Std 1003.3-1991
19 {3}. A set of test methods conforms to this standard if it tests all of the test
20 assertions.

21 **Base Document**

22 The base document for this standard is an unpublished set of test assertions for
23 the X/Open Directory Services API Specification (XDS) {B15}, copyright © 1991 by
24 X/Open Company Limited, United Kingdom. The working group is grateful to
25 X/Open for permission to use their materials.

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

5 **Portable Applications Standards Committee (PASC)**

6 Chair: Jim Isaak
7 Vice Chairs: Hal Jespersen
8 Lorraine Kevra
9 Pete Meier
10 Andrew Twigger
11 Treasurer: Peter Smith
12 Secretary: Shane McCarron

13 **P1224.2 Working Group Officials**

14 Chair: Mark Hazzard
15 Vice Chair: Donna Fisher
16 Editor: Chris Harding
17 Secretary: Donna Fisher

18 **P1224.2 Working Group**

19 Greg Bussiere Edmund Liu Rob Spade
20 Scott Guthery Ted Roberts Alec Tashjian

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

23 Stephen R. Walli *EurOpen Organizational Representative*
24 Ralph Barker *Uniform Organizational Representative*

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

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

3 **Wallace S. Read, Chair** **Andrew G. Salem, Secretary** **Donald C. Loughry, Vice Chair**

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

13 Also included are the following nonvoting IEEE Standards Board liaisons:

14	Satish K. Aggarwal
15	James Beall
16	Richard B. Engelman
17	David E. Soffrin
18	Stanley I. Warshaw

19	Rachel A. Meisel
20	<i>IEEE Standards Project Editor</i>

IEEE Standard for Information Technology—Test Methods for Measuring Conformance to Directory Services C Language Interfaces—Binding for Application Program Interface (API)

Section 1: General

1.1 Scope

This standard defines requirements for test methods for measuring conformance to IEEE Std 1327.2-1993 {8}.

IEEE Std 1327.2-1993 {8} contains a programming language binding specification for IEEE Std 1224.2-1993 {4}, using the C programming language. IEEE Std 1326.2-1993 {6} contains language-independent requirements for test methods for measuring conformance to programming language binding specifications for IEEE Std 1224.2-1993 {4}, such as that contained in IEEE Std 1327.2-1993 {8}. This standard contains C language-specific requirements for the test methods. Taken in conjunction with the requirements imposed by IEEE Std 1326.2-1993, they constitute the requirements that shall be satisfied by test methods used for measuring conformance to IEEE Std 1327.2-1993 {8}.

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 {1} ISO/IEC 9899:1990,¹⁾ *Programming Languages—C*.
- 2 {2} ISO/IEC 9945-1: 1990 (IEEE Std 1003.1: 1990),²⁾ *Information Technology—*
3 *Portable Operating System Interface (POSIX)—Part 1: System Application*
4 *Program Interface (API) [C Language]*.
- 5 {3} IEEE Std 1003.3-1991, *IEEE Standard for Information Technology—Test*
6 *Methods for Measuring Conformance to POSIX*.
- 7 {4} IEEE Std 1224.2-1993, *IEEE Standard for Information Technology—*
8 *Directory Services—Application Program Interface (API) [Language Inde-*
9 *pendent]*.
- 10 {5} IEEE Std 1326-1993, *IEEE Standard for Information Technology—Test*
11 *Methods for Measuring Conformance to Open Systems Interconnection (OSI)*
12 *Abstract Data Manipulation—Application Program Interface (API) [Lan-*
13 *guage Independent]*.
- 14 {6} IEEE Std 1326.2, *IEEE Standard for Information Technology—Test Meth-*
15 *ods for Measuring Conformance to Directory Services—Application Program*
16 *Interface (API) [Language Independent]*.
- 17 {7} IEEE Std 1327-1993, *IEEE Standard for Information Technology—Open*
18 *Systems Interconnection (OSI) Abstract Data Manipulation C Language*
19 *Interfaces—Binding for Application Program Interface (API)*.
- 20 {8} IEEE Std 1327.2-1993, *IEEE Standard for Information Technology—*
21 *Directory Services C Language Interfaces—Binding for Application Program*
22 *Interface (API)*.
- 23 {9} IEEE Std 1328-1993, *IEEE Standard for Information Technology—Test*
24 *Methods for Measuring Conformance to Open Systems Interconnection (OSI)*
25 *Abstract Data Manipulation C Language Interfaces—Binding for Applica-*
26 *tion Program Interface (API)*.

27 1.3 Conformance

28 A set of test methods that conforms to this standard shall conform to IEEE Std
29 1003.3-1991 {3}, with references in IEEE Std 1003.3-1991 {3} to the “POSIX.n test
30 method specification” being interpreted as references to this standard, and refer-
31 ences in IEEE Std 1003.3-1991 {3} to “the POSIX standard for which conformance
32 is being measured” being interpreted as references to IEEE Std 1327.2-1993 {8}.

- 33
- 34 1) ISO/IEC documents can be obtained from the ISO office, 1 Rue de Varembe, Case Postale 56, CH-1211,
35 Genève 20, Switzerland/Suisse.
- 36 2) IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445
37 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

1 In addition to meeting the conformance criteria defined in IEEE Std 1003.3-1991
2 {3}, a set of test methods that conforms to this standard shall test all documenta-
3 tion assertions defined in this standard.

4 NOTE: Conformance to IEEE Std 1003.3-1991 {3} implies that the test methods will test all other
5 assertions defined in this standard.

6 A set of test methods that conforms to this standard shall also conform to IEEE
7 Std 1326.2-1993 {6}.

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 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**).

Function synopses and other extended pieces of C code appear in constant width (courier) font while C language names embedded in ordinary text appear in italic font. Underscores are used to separate words in C language names. The C language names are derived mechanically from the corresponding language-independent names in a manner described in 2.1.4.

A C language function, datatype, or code fragment appears in *italic* font. A function name is indicated by following parentheses [e.g., *ds_abandon()*].

A C language symbolic constant, other than an error name, is surrounded by braces (e.g., {DS_MAX_OUTSTANDING_OPERATIONS}).

A C language symbolic constant for an error name is surrounded by brackets (e.g., [DS_E_ADMIN_LIMIT_EXCEEDED]).

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*
- C language names embedded in ordinary text
- 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***

- 1 (5) The **constant width** (Courier) font is used for
2 — References to terms defined in the X.500 directory standards
3 — Function synopses and other extended pieces of C code

4 **2.1.2 Representation of Strings of Numbers**

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

10 **2.1.3 Language-Independent Conventions**

11 The language-independent specification for this standard is contained in IEEE Std
12 1224.2-1993 {4}. The language-independent specification conventions apply to that
13 standard and to language-independent specification terms used in this standard.
14 These conventions are described in IEEE Std 1224-1993 {B13}.

15 **2.1.4 C Language Binding Conventions**

16 **2.1.4.1 Introduction**

17 IEEE Std 1327.2-1993 {8} specifies C identifiers for all the elements of the inter-
18 face, so that application programs written in C can access the Directory. These ele-
19 ments include function names, typedef names, and constants. All the C identifiers
20 are mechanically derived from the language-independent names as explained
21 below.

22 **2.1.4.2 C Naming Conventions**

23 The interface uses part of the C public namespace for its facilities. All identifiers
24 start with the letters *ds*, *DS*, or *OMP*. More detail about the conventions used is
25 given in Table 2-1. The interface reserves all identifiers starting with the letters
26 *dsP* for private (i.e., internal) use by implementations of the interface. It also
27 reserves *all* identifiers starting with the letters *dsX* or *DSX* for extensions of the
28 interface. If the `<xds.h>` header is included, the application shall not use any
29 identifier starting with these letters.

30 IEEE Std 1224-1993 {B13} uses similar, though not identical, naming conventions.
31 All OSI abstract data manipulation identifiers are prefixed by the letters *OM* or
32 *om*.

Table 2-1 – C Naming Conventions

Item	Prefix
Reserved for implementors	<i>dsP</i>
Reserved for interface extensions	<i>dsX</i>
Reserved for interface extensions	<i>DSX</i>
Reserved for implementors	<i>OMP</i>
Functions	<i>ds_</i>
Error “problem” values	<i>DS_E_</i>
OM class names	<i>DS_C_</i>
OM value length limits	<i>DS_VL_</i>
OM value number limits	<i>DS_VN_</i>
Other constants	<i>DS_</i>
Attribute Type	<i>DS_A_</i>
Object Class	<i>DS_O_</i>

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

- The C function names are identical to the language-independent operation names. Within text, they are italicized and followed by “()”.

Thus

- *ds_receive_result*

becomes

- *ds_receive_result()*

- C function input parameters are identical to the corresponding language-independent argument names. C function output parameters are derived from the argument names by adding “_return” as a suffix. Thus, the output argument

- *operation_status*

becomes

- *operation_status_return*

(Within text, both language-independent argument names and C function parameter names are italicized.)

- The names of constants identifying OM classes are derived from the class names by converting lowercase to uppercase, converting hyphens to underscores, and adding the prefix “DS_C_”.

Thus

- **Read-Result**

becomes

- {DS_C_READ_RESULT}

- 1 — The names of C language constants that denote language-independent
2 specification choice values are derived from the choice value names by
3 converting lowercase to uppercase and adding the prefix “DS_”.
4 Thus
 - 5 — **Default-Context**
6 becomes
7 — {DS_DEFAULT_CONTEXT}
- 8 — Enumeration tags are derived from the name of the corresponding OM syn-
9 tax by adding the prefix “DS_”. Hyphens are converted to underscores, but
10 the case of letters is left unchanged.
11 Thus
 - 12 — **Enum(Limit-Problem)**
13 becomes
14 — *DS_Limit_Problem*
- 15 — For enumeration constants, OM attributes and all other constants except
16 errors, lowercase is converted to uppercase, hyphens are converted to
17 underscores, and the prefix “DS_” is added.
18 Thus
 - 19 — **O-Residential-Person**
20 becomes
21 — {DS_O_RESIDENTIAL_PERSON}
- 22 — Errors are treated as a special case. Constants that are the possible values
23 of the OM attribute **Problem** of a subclass of the OM class **Error** have
24 hyphens converted to underscores, are made entirely uppercase, and are
25 prefixed by “DS_E_”.
26 Thus
 - 27 — alias-dereferencing-problem
28 becomes
29 — [DS_E_ALIAS_DEREFERENCING_PROBLEM]
- 30 — The constants in the “Value Length” and “Value Number” columns of the
31 OM class definition tables are also assigned identifiers. (They have no
32 names in the language-independent specification.) When the upper limit in
33 one of these columns is not “1” (one), it is given a name consisting of the OM
34 attribute name prefixed by “DS_VL_” for value length or “DS_VN_” for
35 value numbers.
- 36 — The sequence of octets for each object identifier is also assigned an identifier
37 for internal use by certain OM macros. These identifiers are all uppercase
38 and are prefixed by “OMP_O_”. See IEEE Std 1224-1993 {B13} for further
39 details on the use of object identifiers.
- 40 Note that hyphens are translated to underscores.

2.1.4.3 Use and Implementation of Interfaces

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

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

These concepts apply in this standard unless explicitly stated otherwise in the C language binding descriptions.

2.1.4.4 Function Return Values

Except for *ds_initialize()*, the return value of a C function is bound to the *Status* result of the language-independent description. Functions shall return a value of type *DS_status*, which is an error indication. If and only if the function succeeds, its value shall be **success**, expressed in the C language by the constant {DS_SUCCESS}. If a function returns a status other than this, then it shall not have updated the return parameters. The value of the status, in this case, shall be an error as described in IEEE Std 1224.2-1993 {4}.

Since the C language does not provide multiple return values, functions shall return all other results by writing them into storage passed by the application program. Any argument that is a pointer to such storage has a name ending with *_return*. For example, the C parameter declaration

— *om_uint * completion_flag_return*

indicates that the function shall return an unsigned integer *Completion-Flag* as a result, so the actual argument to the function must be the address of a suitable variable. This notation allows the reader to distinguish between an input parameter that happens to be a pointer and an output parameter where the “*” is used to simulate the semantics of passing by reference.

2.1.5 Test Methods Conventions

The assertion classification and labeling schemes defined in IEEE Std 1003.3-1991 {3} are used in this standard with the following addition.

This standard contains an assertion for each statement that is derived from the standard to which conformance is being measured, that pertains to the documentation associated with the implementation being tested, and that is true for a conforming implementation. These assertions are called *documentation assertions*. Each documentation assertion is labeled Dnn, where nn is a two digit number that is unique within this standard.

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 of ISO/IEC 9945-1: 1990 {2}.

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 {2}]

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. [IEEE Std 1224.2-1993 {4}]

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 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: 1990 {2}]

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 {2}]

2.2.1.6 should:

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

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

4 **2.2.1.7 supported:** A condition regarding optional functionality.

5 Certain functionality in this standard is optional, but the interfaces to that func-
6 tionality are always required. If the functionality is *supported*, the interfaces work
7 as specified in this standard (except that they do not return the error condition
8 indicated for the unsupported case). If the functionality is not *supported*, the
9 interface shall always return the indication specified for this situation. [ISO/IEC
10 9945-1: 1990 {2}]

11 **2.2.1.8 system documentation:** All documentation provided with an implemen-
12 tation, except the conformance document.

13 Electronically distributed documents for an implementation are considered part of
14 the system documentation. [ISO/IEC 9945-1: 1990 {2}]

15 **2.2.1.9 undefined:** An indication that this standard imposes no portability
16 requirements on an application's use of an indeterminate value or its behavior
17 with erroneous program constructs or erroneous data.

18 Implementations (or other standards) may specify the result of using that value or
19 causing that behavior. An application using such behaviors is using extensions, as
20 defined in 1.3.2.3 of ISO/IEC 9945-1: 1990 {2}.

21 **2.2.1.10 unspecified:** An indication that this standard imposes no portability
22 requirements on applications for correct program constructs or correct data
23 regarding a value or behavior.

24 Implementations (or other standards) may specify the result of using that value or
25 causing that behavior. An application requiring a specific behavior, rather than
26 tolerating any behavior when using that functionality, is using extensions, as
27 defined in 1.3.2.3 of ISO/IEC 9945-1: 1990 {2}.

28 **2.2.2 General Terms**

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

30 **2.2.2.1 abstract class:** An OM class of OM objects of which instances are
31 forbidden. [IEEE Std 1224.2-1993 {4}]

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 {B1}. [IEEE Std 1224.2-1993 {4}]

2.2.2.3 access point: The point at which an abstract service is obtained. [ISO/IEC 9594-2: 1990 {B6}]

2.2.2.4 address: An unambiguous name, label, or number that identifies the location of a particular entity or service. [IEEE Std 1224.2-1993 {4}]

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: 1990 {B6}]

2.2.2.6 alias entry: A directory entry, of object class “alias,” containing information used to provide an alternative name for an object. [IEEE Std 1224.2-1993 {4}]

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. [IEEE Std 1224.2-1993 {4}]

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. [IEEE Std 1224.2-1993 {B13}]

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. [IEEE Std 1224.2-1993 {4}]

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. [IEEE Std 1224.2-1993 {4}]

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

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. [IEEE Std 1224.2-1993 {4}]

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, about the values (or perhaps only the distinguished values) of an entry. [ISO/IEC 9594-2: 1990 {B6}]

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 {3}]

2.2.2.19 basic encoding rules: A specific set of rules used to encode ASN.1 values as strings of octets. [IEEE Std 1224.2-1993 {4}]

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: 1990 {B8}]

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. [IEEE Std 1224.2-1993 {4}]

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). [IEEE Std 1224.2-1993 {4}]

2.2.2.26 concrete class: An OM class of which instances are permitted. [IEEE Std 1224.2-1993 {4}]

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. [IEEE Std 1224.2-1993 {4}]

2.2.2.28 descriptor list: An ordered sequence of descriptors that is used to represent several OM attribute types and values. [IEEE Std 1224.2-1993 {4}]

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: 1990 {B6}]

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: 1990 {B6}]

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: 1990 {B6}]

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: 1990 {B6}]

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: 1990 {B6}]

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: 1990 {B6}]