

**Information technology —
Test methods for measuring
conformance to directory
services C language
interfaces —
Binding for Application
Program Interface (API)**

ICS 35.100.70

National foreword

This British Standard reproduces verbatim ISO/IEC 14395 : 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 vi, pages 1 to 65 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 26585 4

Amendments issued since publication

Amd. No.	Date	Text affected

INTERNATIONAL STANDARD

**ISO/IEC
14395**

First edition
1996-06-01

Information technology — Test methods for measuring conformance to directory services C language interfaces — Binding for Application Program Interface (API)

*Technologies de l'information — Méthodes d'essai pour mesurer la
conformité aux interfaces de langage C des services de l'annuaire —
Liant pour interface de programme d'application (API)*



Reference number
ISO/IEC 14395:1996(E)

AVA 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	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	30
5.2.9 Modify Entry	31
5.2.10 Modify RDN	32
5.2.11 Read	32
5.2.12 Receive Result	33
5.2.13 Remove Entry	34
5.2.14 Search	34
5.2.15 Shutdown	35
5.2.16 Unbind	36
5.2.17 Version	36

5.3	Directory Service Package	37
5.3.1	Object Identifiers	37
5.3.2	Enumeration Types and Constants	39
5.3.3	OM Object Constants	41
5.3.4	Integer Constants	41
5.3.5	Class Constraints	42
5.4	The <xds.h> Header	42
Section 6: Test Assertions—The X.500 Directory Service		43
6.1	Overview	43
6.2	Basic Directory Contents Package	43
6.2.1	Directory Object and Attribute Object Identifiers	43
6.2.2	OM Object and Attribute Type Identifiers	44
6.2.3	Directory Enumeration Types and Constants	45
6.2.4	Integer Constants	46
6.3	Strong Authentication Package	47
6.3.1	Directory Object and Attribute Object Identifiers	47
6.3.2	OM Object and Attribute Type Identifiers	47
6.3.3	Enumeration Types and Constants	48
6.3.4	Integer Constants	48
6.4	MHS Directory User Package	49
6.4.1	Directory Object and Attribute Object Identifiers	49
6.4.2	OM Object and Attribute Type Identifiers	49
6.4.3	Enumeration Types and Constants	50
6.5	Header Files	50
6.5.1	The <xdsbdc.h> Header	50
6.5.2	The <xdssap.h> Header	50
6.5.3	The <xdsmdup.h> header	51
Section 7: Test Assertions—Definitions of Constants		53
7.1	Introduction	53
7.2	Directory Service Package	53
7.2.1	Object and Attribute Type Identifiers	53
7.2.2	Enumeration Types and Constants	53
7.2.3	OM Object Constants	56
7.2.4	Integer Constants	57
7.3	Basic Directory Contents Package	57
7.4	Strong Authentication Package	58
7.5	MHS Directory User Package	58
Annex A (informative) Bibliography		59
Alphabetic Topical Index		61
TABLES		
Table 2-1	– C Naming Conventions	8
Table 2-2	– Identifier Abbreviations	10

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 14395 was prepared by IEEE (as IEEE Std 1328.2-1993) 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 14395, 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.)

The purpose of this International Standard is to define test methods for the C language binding contained in ISO/IEC 14394 {8} for the language-independent specification contained in ISO/IEC 14392 {6} 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 {6} defines, in programming language independent terms, an API to directory services. This API is known as the Directory Services API (DS API). ISO/IEC 14394 {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. ISO/IEC 14392 {6} 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 ISO/IEC 14392 {6} 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.

30 Related Standards

ISO/IEC 14392 {6} 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 {8}. This International Standard applies to test methods for measuring conformance to that programming language binding specification.

ISO/IEC 14393 {7} specifies test methods for the language-independent specification contained in ISO/IEC 14392 {6}. A set of test methods for the C binding to ISO/IEC 14392 {6} must satisfy the requirements of ISO/IEC 14393 {7}, as well as conforming to this International Standard.

The API defined in ISO/IEC 14392 {6} uses the mechanism for OSI abstract data Manipulation (OM) defined in ISO/IEC 14360 {B13}. ISO/IEC 14362 {3} defines the requirements that apply to test methods for measuring conformance to ISO/IEC 14360 {B13}. A C language binding to ISO/IEC 14360 {B13} is defined in ISO/IEC 14364 {4}. ISO/IEC 14366 {5} defines the requirements that apply to test methods for measuring conformance to ISO/IEC 14364 {4}.

IEEE Std 1003.3-1991 {9} defines the general requirements that shall apply to test methods for measuring conformance to POSIX. A set of test methods used to measure conformance to ISO/IEC 14392 {6} must satisfy the requirements of IEEE Standard 1003.3, as well as conforming to this International Standard.

Overview

For each section of ISO/IEC 14394 {8}, this International Standard contains a corresponding section containing test assertions, in accordance with IEEE Std 1003.3-1991 {9}. A set of test methods conforms to this International Standard if it tests all of the test assertions.

This International Standard is based on IEEE Std 1328.2-1993 {B15}, which was prepared by the Namespace and Directory Services Working Group (P1224.2, formerly P1003.17), sponsored by the Portable Applications Standards Committee of the IEEE Computer Society.

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 International Standard defines requirements for test methods for measuring conformance to ISO/IEC 14394 {8}.

ISO/IEC 14394 {8} contains a programming language binding specification for ISO/IEC 14392 {6}, using the C programming language. ISO/IEC 14393 {7} contains language-independent requirements for test methods for measuring conformance to programming language binding specifications for ISO/IEC 14392 {6}, such as that contained in ISO/IEC 14394 {8}. This International Standard contains C language-specific requirements for the test methods. Taken in conjunction with the requirements imposed by ISO/IEC 14393 they constitute the requirements that shall be satisfied by test methods used for measuring conformance to ISO/IEC 14394 {8}.

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 9899:1990,¹⁾ *Programming languages—C*.
- {2} ISO/IEC 9945-1: 1990²⁾ (IEEE Std 1003.1: 1990),³⁾ *Information technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C language]*.
- {3} ISO/IEC 14362: 1996, *Information technology—Test methods for measuring conformance to Open Systems Interconnection (OSI) abstract data manipulation—Application Program Interface (API) [Language independent]*.
- {4} ISO/IEC 14364: 1996, *Information technology—Open Systems Interconnection (OSI) abstract data manipulation C language interfaces—Binding for Application Program Interface (API)*.
- {5} ISO/IEC 14366: 1996, *Information technology—Test methods for measuring conformance to Open Systems Interconnection (OSI) abstract data manipulation C language interfaces—Binding for Application Program Interface (API)*.
- {6} ISO/IEC 14392: 1996, *Information technology—Directory services—Application Program Interface (API) [Language independent]*.
- {7} ISO/IEC 14393: 1996, *Information technology—Test methods for measuring conformance to directory services—Application Program Interface (API) [Language independent]*.
- {8} ISO/IEC 14394: 1996, *Information Technology—Directory services C language interfaces—Binding for Application Program Interface (API)*.
- {9} IEEE Std 1003.3-1991, *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX*.

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.

2) ISO/IEC 9945-1: 1990 is currently under revision.

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

1.3 Conformance

A set of test methods that conforms to this International Standard shall conform to IEEE Std 1003.3-1991 {9}, with references in IEEE Std 1003.3-1991 {9} to the “POSIX.n test method specification” being interpreted as references to this International Standard, and references in IEEE Std 1003.3-1991 {9} to “the POSIX standard for which conformance is being measured” being interpreted as references to ISO/IEC 14394 {8}.

In addition to meeting the conformance criteria defined in IEEE Std 1003.3-1991 {9}, 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 {9} implies that the test methods will test all other assertions defined in this International Standard.

A set of test methods that conforms to this International Standard shall also conform to ISO/IEC 14393 {7}.

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, 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 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*
- 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***

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

- References to terms defined in the X.500 directory standards
- Function synopses and other extended pieces of C code.

2.1.2 Representation of Strings of Numbers

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

2.1.3 Language-Independent Conventions

The language-independent specification for this International Standard is contained in ISO/IEC 14392 {6}. The language-independent specification conventions apply to that standard and to language-independent specification terms used in this International Standard. These conventions are described in ISO/IEC 14360 {B13}.

2.1.4 C Language Binding Conventions

2.1.4.1 Introduction

ISO/IEC 14394 {8} specifies C identifiers for all the elements of the interface, so that application programs written in C can access the Directory. These elements include function names, typedef names, and constants. All the C identifiers are mechanically derived from the language-independent names as explained below.

2.1.4.2 C Naming Conventions

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

ISO/IEC 14360 {B13} uses similar, though not identical, naming conventions. All OSI abstract data manipulation identifiers are prefixed by the letters *OM* or *om*.

The C identifiers are derived from the language-independent names used throughout this International 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()`

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>

- 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}

- The names of C language constants that denote language-independent specification choice values are derived from the choice value names by converting lowercase to uppercase and adding the prefix “*DS_*”.

Thus

— **Default-Context**
becomes
— {DS_DEFAULT_CONTEXT}

- Enumeration tags are derived from the name of the corresponding OM syntax by adding the prefix “*DS_*”. Hyphens are converted to underscores, but the case of letters is left unchanged.

Thus

— **Enum(Limit-Problem)**
becomes
— *DS_Limit_Problem*

- 151 — For enumeration constants, OM attributes and all other constants except
152 errors, lowercase is converted to uppercase, hyphens are converted to under-
153 scores, and the prefix “*DS_*” is added.
Thus
- 154 — **O-Residential-Person**
155 becomes
156 — {DS_O_RESIDENTIAL_PERSON}
- 157 — Errors are treated as a special case. Constants that are the possible values
158 of the OM attribute **Problem** of a subclass of the OM class **Error** have
159 hyphens converted to underscores, are made entirely uppercase, and are
160 prefixed by “*DS_E_*”.
161 Thus
- 162 — alias-dereferencing-problem
163 becomes
164 — [DS_E_ALIAS_DEREFERENCING_PROBLEM]
- 165 — The constants in the “Value Length” and “Value Number” columns of the
166 OM class definition tables are also assigned identifiers. (They have no
167 names in the language-independent specification.) When the upper limit in
168 one of these columns is not “1” (one), it is given a name consisting of the OM
169 attribute name prefixed by “*DS_VL_*” for value length or “*DS_VN_*” for
170 value numbers.
- 171 — The sequence of octets for each object identifier is also assigned an identifier
172 for internal use by certain OM macros. These identifiers are all uppercase
173 and are prefixed by “*OMP_O_*”. See ISO/IEC 14360 {B13} for further details
174 on the use of object identifiers.
- 175 — In some cases, the transformations described above result in identifiers that
176 are more than 32 characters in length. The ISO C Standard {1} only requires
177 identifiers to be unique within the first 32 characters. The abbreviations
178 listed in Table 2-2 are applied to substrings of identifiers to reduce their
179 length.
- 180 Note that hyphens are translated to underscores.

181 2.1.4.3 Use and Implementation of Interfaces

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

185 Any function declared in a header may be implemented as a macro defined in the
186 header, so a library function should not be declared explicitly if its header is
187 included. Any macro definition of a function can be suppressed locally by enclosing
188 the name of the function in parentheses, because the name is not then followed by
189 the left parenthesis that indicates expansion of a macro function name. For the
190 same syntactic reason, it is permitted to take the address of a library function even
191 if it is also defined as a macro. The use of *#undef* to remove any macro definition
192 will also ensure that the identifier refers to an actual function. Any invocation of a
193 library function that is implemented as a macro shall expand to code that evalu-
194 ates each of its arguments exactly once, fully protected by parentheses where
necessary, so it is generally safe to use arbitrary expressions as arguments.

2.1 Conventions

Table 2-2 – Identifier Abbreviations

Identifier	Abbreviation	Identifier	Abbreviation
ADMINISTRATIVE	ADMIN	NUMBER	NBR
AGENT	AG	OFFICE	OFF
APPLICATION	APPLIC	OPTIONAL	OPT
AUTOMATIC	AUTO	ORGANIZATIONAL	ORG
CERTIFICATE	CERT	ORGANIZATION	ORG
CRITICAL	CRIT	PERMISSION	PERM
DELIVERY	DELIV	PHYSICAL	PHYS
DESTINATION	DEST	PREFERRED	PREF
EXTENSION	EXT	PROHIBITED	PROHIB
EXTENSIONS	EXT	PROVINCE	PROV
IDENTIFIER	IDENT	QUALIFIER	QUAL
IDENTIFIER	IDENT	REFERENCE	REF
INAPPROPRIATE	INAPPROP	REVOCATION	REVOC
INFORMATION	INFO	SUPPORTED	SUP
INTERNATIONAL	INTERNAT	SUPPORTED	SUPPORT
MODIFICATION	MOD	TERMINAL	TERM

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 International 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 ISO/IEC 14392 [6].

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

236 The assertion classification and labeling schemes defined in IEEE Std 1003.3-1991
237 {9} are used in this International Standard with the following addition.

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

2.2 Definitions

2.2.1 Terminology

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

247 **2.2.1.1 conformance document:** A document provided by an implementor that
248 contains implementation details as described in 1.3.1.2 of ISO/IEC 9945-1 {2}.

249 **2.2.1.2 implementation defined:** An indication that the implementation shall
250 define and document the requirements for correct program constructs and correct
251 data of a value or behavior. [ISO/IEC 9945-1 {2}]

252 **2.2.1.3 locally administered:** An indication that the configuration is not con-
253 sistent across all systems and that the administrator of each system shall provide
254 documentation of its behavior. [ISO/IEC 14392 {6}]

255 **2.2.1.4 may:** An indication of an optional feature.

256 With respect to implementations, the word *may* is to be interpreted as an optional
257 feature that is not required in this International Standard, but that can be pro-
258 vided. With respect to Strictly Conforming Applications, the word *may* means that
259 the optional feature shall not be used. [ISO/IEC 9945-1 {2}]

260 **2.2.1.5 shall:** An indication of a requirement on the implementation or on Strictly
261 Conforming Applications, where appropriate. [ISO/IEC 9945-1 {2}]

262 **2.2.1.6 should:**

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

265 (2) With respect to applications, an indication of a recommended pro-
266 gramming practice for applications and a requirement for Strictly
267 Conforming Applications. [ISO/IEC 9945-1 {2}]

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 {2}]

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 {2}]

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 {2}.

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 {2}.

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 {6}]

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 {B1}. [ISO/IEC 14392 {6}]

2.2.2.3 access point: The point at which an abstract service is obtained. [ISO/IEC 9594-2 {B5}]

2.2.2.4 address: An unambiguous name, label, or number that identifies the location of a particular entity or service. [ISO/IEC 14392 {6}]

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 {B5}]

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 {6}]

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 {6}]

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 {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. [ISO/IEC 14392 {6}]

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 {6}]

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 {6}]

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 {B5}]

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 {9}]

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 {6}]

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 {B7}]

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 {6}]

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 {6}]

2.2.2.26 concrete class: An OM class of which instances are permitted. [ISO/IEC 14392 {6}]

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 {6}]

- 363 **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 {6}]
- 364 **2.2.2.29 directory:** A repository of information about objects that provides direc-
365 tory services to its users, which allows its users access to the information. [ISO/IEC
366 9594-2 {B5}]
- 367 **2.2.2.30 directory attribute [attribute]:** The information of a particular type
368 concerning an object and appearing in an entry describing the object in the DIB.
369 [ISO/IEC 9594-2 {B5}]
- 370 **2.2.2.31 directory attribute type [attribute type]:** That component of a *direc-*
371 *tory attribute* that indicates the class of information given by that attribute.
372 It is an *object identifier*, and so completely unique. [ISO/IEC 9594-2 {B5}]
- 373 **2.2.2.32 directory attribute value [attribute value]:** A particular instance of
374 the class of information indicated by a directory attribute type. [ISO/IEC 9594-2
375 {B5}]
- 376 **2.2.2.33 directory class:** See *object class* in 2.2.2.58.
- 377 **2.2.2.34 directory information base:** The complete set of information to which
378 the directory provides access and that includes all of the pieces of information that
379 can be read or manipulated using the operations of the directory. [ISO/IEC 9594-2
380 {B5}]
- 381 **2.2.2.35 directory information tree:** The directory information base, con-
382 sidered as a tree, whose vertices (other than the root) are the directory entries.
383 [ISO/IEC 9594-2 {B5}]
- 384 **2.2.2.36 directory object:** Anything in some “world,” generally the world of
385 telecommunications and information processing or some part thereof, that is
386 identifiable (can be named) and that it is of interest to hold information on in the
387 directory information base. [ISO/IEC 9594-2 {B5}]
- 388 **2.2.2.37 directory operation:** Processing performed within the directory to pro-
389 vide a service, such as a read directory operation; it is given some arguments as
390 input, performs some processing, and returns some results.
391 An application process causes a directory operation to be performed by invoking an
392 interface operation. [ISO/IEC 14392 {6}]
- 393 **2.2.2.38 directory syntax:** See *attribute syntax* in 2.2.2.13.
- 394 **2.2.2.39 directory system agent:** An OSI application process that is part of the
395 Directory. [ISO/IEC 9594-2 {B5}]