

IEEE Standard for Information Technology— Open Systems Interconnection (OSI) Abstract Data Manipulation—Application Program Interface (API) [Language Independent]

Sponsor
Portable Applications Standards Committee
of the
IEEE Computer Society

Approved March 18, 1993
IEEE Standards Board

Abstract: A general-purpose application program interface (API) for the creation, examination, modification, and deletion of potentially complex Open Systems Interconnection (OSI) information objects is defined in terms that are independent of any particular programming language. The API provides tools for manipulating objects defined using Abstract Syntax Notation One (ASN.1), shielding the application programmer from much of the complexity of ASN.1. It is designed to be used in conjunction with, but is otherwise independent of, application-specific APIs for OSI.

Keywords: abstract syntax notation, API, application portability, ASN.1, language independent, object management, open systems, open systems interconnection, OSI

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

Copyright © 1993 by the
Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1993
Printed in the United States of America

ISBN 1-55937-301-6

*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.*

THIS PAGE WAS
BLANK IN THE ORIGINAL

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

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

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

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

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

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

IEEE Standards documents 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	2
1.3 Conformance	2
1.4 Test Methods	5
Section 2: Terminology and General Requirements	7
2.1 Conventions	7
2.2 Definitions	10
Section 3: Overview	17
3.1 Purpose	17
3.2 Motivation	18
Section 4: Information Architecture	21
4.1 Introduction	21
4.2 Objects	21
4.3 Object Attributes	22
4.4 Classes	23
4.5 Class Definitions	24
4.6 Packages	25
4.7 Package Definitions	25
4.8 Workspaces	26
4.9 Storage Management	26
Section 5: Information Syntaxes	29
5.1 Introduction	29
5.2 Syntax Templates	29
5.3 Syntaxes	29
5.4 Strings	30
5.4.1 String Lengths	30
5.5 Relationship to ASN.1 Simple Types	31
5.6 Relationship to ASN.1 Useful Types	32
5.7 Relationship to ASN.1 Character String Types	32
5.8 Relationship to ASN.1 Type Constructors	32
Section 6: Service Interface	35
6.1 Introduction	35
6.2 Datatypes	35

6.2.1	Signed Integers	35
6.2.2	Unsigned Integers	35
6.2.3	Boolean	36
6.2.4	Enumeration	36
6.2.5	Exclusions	37
6.2.6	Integer	37
6.2.7	Modification	38
6.2.8	Object	38
6.2.9	Object Identifier	38
6.2.10	Private Object	39
6.2.11	Public Object	39
6.2.12	Service-Generated Public Object	40
6.2.13	Client-Generated Public Object	40
6.2.14	Attribute Set	40
6.2.15	Attribute	41
6.2.16	Value	41
6.2.17	Value Data	41
6.2.18	Null Value	42
6.2.19	String Value	42
6.2.20	String Flags	43
6.2.21	String Choice	43
6.2.22	Return Code	44
6.2.23	String	44
6.2.24	Type	45
6.2.25	Type List	45
6.2.26	Value Length	45
6.2.27	Value Position	46
6.2.28	Workspace	46
6.3	Operations	47
6.3.1	Copy	49
6.3.2	Copy Value	50
6.3.3	Create	51
6.3.4	Decode	52
6.3.5	Delete	53
6.3.6	Encode	54
6.3.7	Get	55
6.3.8	Instance	58
6.3.9	Put	59
6.3.10	Read	62
6.3.11	Remove	63
6.3.12	Write	64
6.4	Return Codes	66
Section 7: Object Management Package		71
7.1	Class Hierarchy	71
7.2	Class Definitions	71
7.2.1	Encoding	71
7.2.2	External	72
7.2.3	Object	74

Section 8: Definitions of Constants	75
8.1 Introduction	75
8.2 Object Management API	75
8.3 Object Management Package	75
Annex A (informative) Bibliography	77
Alphabetic Topical Index	79

FIGURES

Figure 3-1 – Conceptual Model of Object Management	18
Figure 4-1 – Structure of an Object	22

TABLES

Table 5-1 – String Syntax Identifiers	31
Table 5-2 – Syntax for ASN.1 Simple Types	31
Table 5-3 – Syntaxes for ASN.1 Useful Types	32
Table 5-4 – Syntaxes for ASN.1 Character String Types	32
Table 5-5 – Syntaxes for ASN.1 Type Constructors	33
Table 6-1 – Service Interface Operations	47
Table 6-2 – Service Interface Return Codes	67
Table 7-1 – Attributes Specific to Encoding	72
Table 7-2 – Attributes Specific to External	73
Table 7-3 – Attributes Specific to Object	74
Table 8-1 – Class Object Identifier Constant Suffixes	76
Table 8-2 – Attribute Type Identifiers	76

Introduction

(This introduction is not a normative part of IEEE Std 1224-1993, IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation—Application Program Interface (API) [Language Independent], but is included for information only.)

Purpose

The purpose of this standard is to define a general-purpose OSI Abstract Data Manipulation (OM) Application Programming Interface (API) in terms that are independent of any particular programming language. The OM interface is designed to be used with application-specific APIs that provide OSI services to allow the transfer of Abstract Syntax Notation One (ASN.1) protocol elements in an application-independent fashion.

Related Standards

This standard is intended to provide the basis for the definition of programming language bindings to which implementations and applications can conform. Such a language binding, for the C programming language, is contained in IEEE Std 1327-1993 [B6].

This standard, and the language bindings derived from it, are intended to be used in the definition of application-specific APIs that provide OSI services, such as the X.400-based API defined in IEEE Std 1224.1-1993 [B4] and the API to directory services defined in IEEE Std 1224.2-1993 [B5].

IEEE Std 1326-1993 [4] contains a set of requirements to be satisfied by test methods for measuring conformance to this standard. They are stated in terms that are independent of any particular programming language, and they apply to test methods for measuring conformance to all standards defining programming language bindings to this standard.

Overview

This standard defines an information architecture that addresses objects that are based on ASN.1. By providing tools for manipulating ASN.1 objects, the OM interface shields the application programmer from much of the complexity of encoding and decoding ASN.1 elements using the ASN.1 Basic Encoding Rules (BER).

The Object Management API provides a platform on which a range of application-specific APIs can be built. An application program must format its data into objects and then submit (or retrieve) these objects using programmatic “calls” that are standardized by the OM interface.

1 The OM API consists of the syntactical definition of an OM object and of the
2 operations that an application can invoke to manipulate instances of OM objects.

3 The OM API presents to the programmer a uniform model for information based
4 on the concept of classes. A class is described by a collection of OM attributes; each
5 attribute consists of one or more values, each of which is characterized by a type
6 and an OM syntax (e.g., Integer, Boolean, and String).

7 The representation hiding that the OM interface provides is not by itself fully ade-
8 quate to meet the needs of environments supporting several application-specific
9 APIs (e.g., where an X.400 Application API and a Directory Services API can co-
10 exist in the same run-time environment). The different APIs may impose differ-
11 ent, even conflicting, requirements on the internal representations of objects, and
12 they might be implemented by different vendors.

13 However, the OM interface does allow any number of OM interface implementa-
14 tions to coexist, each representing objects differently. This is accomplished by
15 means of workspaces.

16 **Acknowledgments**

17 The base document for this standard is the OSI-Abstract-Data-Manipulation API
18 (XOM) (B10), Copyright © 1989–1990 by X.400 API Association, United States of
19 America, and by X/Open Company Limited, United Kingdom. The IEEE is grate-
20 ful to both X/Open and the X.400 API Association for permission to use their
21 materials.

22 **Related Standards Activities**

23 The following areas are under active consideration at this time, or are expected to
24 become active in the near future, concerning standards for application APIs that
25 use the mechanism defined in this standard. Similar efforts can be anticipated in
26 the future.¹⁾

- 27 (1) X.400-based message handling
- 28 (2) Directory services
- 29 (3) FTAM API
- 30 (4) Verification testing methods
- 31 (5) Network interface facilities
- 32 (6) System administration

33
34 1) A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from
35 the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903, USA; Telephone:
36 +1 202 371-0101; FAX: +1 202 728-9614.

- 1 Extensions are approved as “amendments” or “revisions” to this document, follow-
2 ing IEEE Standards procedures.
- 3 Approved amendments are published separately and distributed with orders from
4 the IEEE for this document until the full document is reprinted and such amend-
5 ments are incorporated in their proper positions.
- 6 If you have any question about the completeness of your version, you may contact
7 the IEEE Standards Department [+1 (908) 562-3800] to determine what supple-
8 ments have been published.
- 9 If you have an interest in participating in the working groups addressing these
10 issues, please send your name, address, and phone number to the Secretary, IEEE
11 Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box
12 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA, and ask to have this for-
13 warded to the chairperson of the appropriate PASC working group.
14

This standard was prepared by the P1224 Working Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the P1224 Working Group was as follows:

Portable Applications Standards Committee (PASC)

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

P1224 Working Group Officials

Chair: Steve Trus
Vice Chair: Tim Carter
Editor/Secretary: Graham Jack

Working Group

Greg Bussiere
Curt Campbell
Tony Cincotta
Iain Devine
Chris Harding
Mark Hazzard
Khiem Ho

Petr Janeček
Mark Karan
Eva Kuiper
Edmun Liu
Kevin Montgomery
Ed Owens
Pierre Perrot

Ted Roberts
Enzo Signore
Daniella Sirocchi
Rob Spade
Alec Tashjian
Andrew Twigger
Bruce Weiner

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

Ralph Barker *Uniform Organizational Representative*
Petr Janeček *X/Open Organizational Representative*

David Athersych	Derek Kaufman	Joseph Ramus
Ralph Barker	Norman Kincl	Robert Robillard
Peter Beaman	Joseph King	Wolfgang Schwabl
Andy Bihain	Stephen Layne	Richard Seibel
Mack W. Bishop	Joachim Loitzenbauer	Dan Shia
Tim Carter	Steven Lucas	Jim Sides
Andy Cheese	Joseph F. P. Luhukay	John P. Slone
Kilnam Chon	Roger Martin	James G. Tanner
Ian Devine	Joberto S. B. Martins	Alec Tashjian
Philip H. Enslow	John S. Morris	Ravi Tavakley
K. Frimpong-Ansah	Andrew L. Nicholson	Donn S. Terry
Michel Gien	Jan Osterweil	Trevor J. Thompson
Steve Griesmer	Edward Owens	Mark-Rene Uchida
Scott Guthery	Ramesh Pabbati	Paul Wanish
Charles Hammons	Prayoon Phathayakorn	Bruce Weiner
C. J. Harding	Dave Plauser	Les Wibberley
Mark Hazzard	Udo W. Pooch	Peter Wishart
James Hegerty	Scott E. Preece	Oren Yuen
Niraj Jain	Paul Rabin	George R. Zerdian
Hal Jespersen	Steve Rago	Jason Zions

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

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

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

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
David E. Soffrin
Stanley I. Warshaw

Mary Lynne Nielsen
IEEE Standards Project Editor

IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation— Application Program Interface (API) [Language Independent]

1 Section 1: General

2 1.1 Scope

3 This standard defines a standard interface supporting the manipulation of com-
4 plex arguments and parameters used by X.400 and Directory Services APIs. The
5 interface supports manipulation of abstract data defined in ASN.1 and is for use in
6 conjunction with, but is otherwise independent of, the X.400 and Directory Ser-
7 vices APIs.

8 An application shall be able to link and use multiple implementations of this API.

9 This standard provides a language-independent specification of an interface and
10 environment to support application portability at the source-code level. It is
11 intended to be used by application developers, system implementors, test method
12 writers, and users.

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

1.2 Normative References

The following standards contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

- [1] ISO/IEC 8824: 1990¹⁾ (CCITT X.208: 1988²⁾, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)*.
- [2] ISO/IEC 8825: 1990 (CCITT X.209: 1988), *Information technology—Open Systems Interconnection—Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*.
- [3] ISO/IEC 9594-8: 1990 (CCITT X.509: 1988), *Information technology—Open Systems Interconnection—The Directory—Part 8: Authentication framework*.
- [4] IEEE Std 1326-1993,³⁾ *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to Open Systems Interconnection (OSI) Abstract Data Manipulation—Application Program Interface (API) [Language Independent]*.

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

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

- (1) The implementation shall support all required behavior defined in this standard.
- (2) The implementation shall support all required interfaces defined in the programming language binding specification. Those interfaces shall support the behavior described in this standard and in the programming language specification.

1) ISO/IEC documents can be obtained from the ISO office, 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

2) CCITT documents can be obtained from the CCITT General Secretariat, International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

3) IEEE documents can be obtained from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ, 08855-1331, USA.

- 1 (3) The implementation may provide additional functions or facilities not
2 required by this standard or by the programming language binding
3 specification. Each such nonstandard extension shall be identified as
4 such in the system documentation. Nonstandard extensions, when used,
5 may change the behavior of functions or facilities defined by this standard
6 or by the programming language binding specification. The conformance
7 document shall define an environment in which an application can be run
8 with the behavior specified by this standard and the programming lan-
9 guage binding specification. In no case shall such an environment require
10 modification of a Strictly Conforming Application.
- 11 (4) The implementation shall comprise one or more workspaces (see 4.8),
12 which shall support the OM package (see Section 7).
- 13 (5) In its implementation of the Encode and Decode operations, the service
14 shall support the Basic Encoding Rules (BER) from ISO/IEC 8825-1990
15 [2].⁴⁾

16 1.3.1.2 Documentation

17 A conformance document with the following information shall be available for an
18 implementation claiming conformance to a programming language binding
19 specification for this standard. The conformance document shall be in two parts.
20 The first part shall have the same structure as this standard, with the information
21 presented in the appropriately numbered sections, clauses, and subclauses. The
22 second part shall have the same structure as the programming language binding
23 specification, with the information presented in the appropriately numbered sec-
24 tions, clauses, and subclauses. The conformance document shall not contain infor-
25 mation about extended features or capabilities outside the scope of this standard
26 and the programming language binding specification.

27 The conformance document shall identify the programming language binding
28 specification to which the implementation conforms.

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

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

35 The conformance document shall describe the behavior of the implementation for
36 all implementation-defined features defined in this standard and in the program-
37 ming language binding specification. This requirement shall be met by listing
38 these features and providing either a specific reference to the system

39 _____
40 4) The numbers in curly brackets correspond to those of the references in 1.2.

documentation or providing full syntax and semantics of these features. The conformance document may specify the behavior of the implementation for those features where this standard or the programming language binding specification states that implementations may vary or where features are identified as undefined or unspecified.

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

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

The system documentation should also contain the information found in the conformance document.

1.3.1.3 Conforming Implementation Options

The following aspects of the behavior of the service are implementation defined:

- (1) The local character set representation and the precise mappings between it and the various string syntaxes.
- (2) The length of the longest string that the `om_get` operation will return. This number is no less than 1024 octets.
- (3) Whether the service reports an exception if an object supplied to it as an argument is not minimally consistent.
- (4) Whether the interface operations are atomic.

1.3.2 Application Conformance

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

1.3.2.1 Strictly Conforming Application

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

1 1.3.2.2 Conforming Application

2 There is only one type of Conforming Application.

3 1.3.2.2.1 ISO/IEC Conforming Application

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

11 1.3.2.3 Conforming Application Using Extensions

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

18 1.4 Test Methods

19 Any measurement of conformance to a programming language binding
20 specification for this standard shall be performed using test methods that conform
21 to IEEE Std 1326-1993 (4) and to any additional requirements that may be
22 imposed by the programming language binding specification.

1 Section 2: Terminology and General Requirements

2 2.1 Conventions

3 2.1.1 General and Typographical Conventions

4 Language-independent concrete OM class names, OM attribute names, and OM
5 attribute value names are spelled with hyphens between words. The first letters
6 of language-independent OM class and OM attribute names are capitalized (e.g.,
7 **Arbitrary-Encoding**).

8 Language-independent datatype, operation, argument, and error names are lower-
9 cased and are spelled with underscores between words (e.g., `om_get`).

10 The use of fonts in this standard is as follows.

- 11 — The Helvetica font is used for:
 - 12 — Language-independent operation names, such as `om_copy_value`
 - 13 — Language-independent datatype names, such as `om_exclusions_type`
 - 14 — Language-independent error names, such as `no_such_class`
- 15 — The *italic* font is used for:
 - 16 — Language-independent operation arguments, such as
17 `source_value_position`
 - 18 — The introduction of important terms
 - 19 — Cross-references in 2.2
- 20 — The **bold** font is used for:
 - 21 — Language-independent concrete OM class names, such as **External**
 - 22 — Language-independent OM attribute names, such as **Arbitrary-**
23 **Encoding**
 - 24 — Language-independent OM attribute values, such as **insert-at-certain-**
25 **point**
- 26 — The ***bold italic*** font is used for:
 - 27 — Language-independent abstract OM class names, such as ***Object***

- 1 — The constant width (Courier) font is used for:
2 — References to terms defined in ISO/IEC 8824: 1990 {1}, ISO/IEC 8825:
3 1990 {2}, and ISO/IEC 9594-8: 1990 {3}.

4 2.1.2 Language-Independent Conventions

5 NOTE: In the absence of a normative notation for language-independent specifications, these con-
6 ventions are based upon a draft of the IEEE P1003.1LIS conventions {B2}.

7 This subclause defines the notation used within this standard to specify the infor-
8 mation that is communicated through interfaces to the defined services. The nota-
9 tion assumes a model of an abstract procedural interface, although language bind-
10 ings to this standard may take other forms.

11 The key elements of the notation are abstract data *datatype* definitions, *name*
12 definitions, and *operation* definitions. The semantics of each of these elements is
13 defined in the General Terms subclause, 2.2.2.

14 NOTE: The notations specified are supplemented by additional descriptions using other notational
15 schemes (including natural language). For instance, for the specification of each language-
16 independent operation, the notation described below is used in the **Synopsis** subclause, while the
17 semantics are described in the **Description** subclause using natural language.

18 In particular, in **Description** subclauses, a notation is used for integer value ranges where a “(” or
19 “)” indicates an *exclusive* range limit, whereas a “[” or “]” indicates an *inclusive* range limit. Thus
20 the range $(0,2^{16})$ specifies integers from 0 to $65536 - 1$ inclusive.

21 2.1.2.1 Notation

22 A modified backus-naur form (BNF) notation is used in this subclause. The char-
23 acters “:”, “=”, “<”, “>”, “|”, “[”, and “]” are always part of this notation and are
24 never used in terminal strings.

25 Nonterminal elements consist of a “<” character followed by an identifier followed
26 by a “>” character. Each production consists of a nonterminal followed by the
27 characters “::=” followed by a right-hand-side expression. In a right-hand-side
28 expression, the “|” character separates alternate elements, and the “[” and “]”
29 characters delimit optional elements.

1 **2.1.2.2 Identifiers**

```

2      <identifier> ::= <initial-char> <identifier-body>
3      <initial-char> ::= <uppercase-letter> | <lowercase-letter>
4      <identifier-body> ::= <body-char> { <identifier-body> }
5      <body-char> ::= <uppercase-letter> | <lowercase-letter> |
6                      <digit> | <separator>
7      <lowercase-letter> ::= a | b | c | d | e | f | g | h | i |
8                      j | k | l | m | n | o | p | q | r | s | t |
9                      u | v | w | x | y | z
10     <uppercase-letter> ::= A | B | C | D | E | F | G | H | I |
11                      J | K | L | M | N | O | P | Q | R | S | T |
12                      U | V | W | X | Y | Z
13     <digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
14     <separator> ::= - | _

```

15 **2.1.2.3 Numbers**

```

16     <number> ::= <digit> | <digit> <number>
17     <signed-number> ::= { - } <number>

```

18 **2.1.2.4 Datatypes**

19 The following notation is used to specify a datatype:

```

20     Datatype:          <datatype-name>
21     Definition:       <datatype>
22     Description:      ...

```

23 where

```

24     <datatype> ::= <datatype-name> | <datatype-definition>
25     <datatype-name> ::= <identifier>
26     <datatype-definition> ::= <atomic-type> | <constructed-type>
27     <atomic-type> ::= boolean_type | <opaque-type> |
28                     <state-type> | <ordinal-type> | <integer-type>
29     <opaque-type> ::= opaque [ <size> ]
30     <state-type> ::= state [ <state-list> ]
31     <state-list> ::= <state-value> { , <state-list> }
32     <state-value> ::= <identifier>
33     <ordinal-type> ::= ordinal_type [ <size> ]
34     <size> ::= <number>
35     <integer-type> ::= integer_type [ <minimum> , <maximum> ]
36     <minimum> ::= <signed-number>
37     <maximum> ::= <signed-number>
38     <constructed-type> ::= <array-type> | <record-type> |
39                     <set-type> | <choice-type> | <set-type>
40     <array-type> ::= array <index-type> of <base-type>
41     <index-type> ::= <ordinal-type> | <integer-type> | <state-type>
42     <base-type> ::= <datatype>
43     <record-type> ::= record [ <member-list> ]
44     <choice-type> ::= choice [ <member-list> ]
45     <set-type> ::= set [ <datatype> ]

```

```

1  <list-type> ::= list [ <datatype> ]
2  <member-list> ::= <member-definition> { , <member-list> }
3  <member-definition> ::= <member-name> : <member-type>
4  <member-name> ::= <identifier>
5  <member-type> ::= <datatype>
6  Any of the values within square brackets may be listed as <implementation
7  defined>; see 2.2.1.2 for the requirements imposed by this notation.

```

8 2.1.2.5 Names

9 The following notation is used to specify a name:

```

10      Name:           <identifier>
11      Datatype:       <datatype-definition>
12      Description:    ...

```

13 2.1.2.6 Operations

14 The following notation is used to specify a operation:

```

15      Operation:      <identifier>
16      Status Datatype: <datatype>
17      Input:          <parameter-name> : <datatype>
18                   <parameter-name> : <datatype>
19                   ...
20      Output:         <parameter-name> : <datatype>
21                   <parameter-name> : <datatype>
22                   ...

```

23 where

```

24      <parameter-name> ::= <identifier>

```

25 2.2 Definitions

26 2.2.1 Terminology

27 For the purposes of this standard, the following definitions apply:

28 **2.2.1.1 conformance document:** A document provided by an implementor that
29 contains implementation details as described in 1.3.1.2. [ISO/IEC 9945-1: 1990
30 {B1}]

31 **2.2.1.2 implementation defined:** An indication that the implementation shall
32 define and document the requirements for correct program constructs and correct
33 data of a value or behavior. [ISO/IEC 9945-1: 1990 {B1}]

- 1 **2.2.1.3 may:** An indication of an optional feature.
- 2 With respect to implementations, the word *may* is to be interpreted as an optional
3 feature that is not required in this standard, but that can be provided. With
4 respect to Strictly Conforming Applications, the word *may* means that the optional
5 feature shall not be used. [ISO/IEC 9945-1:1990 (B1)]
- 6 **2.2.1.4 shall:** An indication of a requirement on the implementation or on
7 Strictly Conforming Applications, where appropriate. [ISO/IEC 9945-1:1990 (B1)]
- 8 **2.2.1.5 should:**
- 9 (1) With respect to implementations, an indication of an implementation rec-
10 ommendation, but not a requirement.
- 11 (2) With respect to applications, an indication of a recommended program-
12 ming practice for applications and a requirement for Strictly Conforming
13 Applications. [ISO/IEC 9945-1:1990 (B1)]
- 14 **2.2.1.6 supported:** A condition regarding optional functionality.
- 15 Certain functionality in this standard is optional, but the interfaces to that func-
16 tionality are always required. If the functionality is *supported*, the interfaces
17 work as specified by this standard (except that they do not return the error condi-
18 tion indicated for the unsupported case). If the functionality is not *supported*, the
19 interface shall always return the indication specified for this situation. [ISO/IEC
20 9945-1:1990 (B1)]
- 21 **2.2.1.7 system documentation:** All documentation provided with an implemen-
22 tation, except the conformance document.
- 23 Electronically distributed documents for an implementation are considered part of
24 the system documentation. [ISO/IEC 9945-1:1990 (B1)]
- 25 **2.2.1.8 undefined:** An indication that this standard imposes no portability
26 requirements on an application's use of an indeterminate value or its behavior
27 with erroneous program constructs or erroneous data.
- 28 Implementations (or other standards) may specify the result of using that value or
29 causing that behavior. An application using such behaviors is using extensions, as
30 defined in 1.3.2.3. [ISO/IEC 9945-1:1990 (B1)]
- 31 **2.2.1.9 unspecified:** An indication that this standard imposes no portability
32 requirements on applications for correct program constructs or correct data
33 regarding a value or behavior.
- 34 Implementations (or other standards) may specify the result of using that value or
35 causing that behavior. An application requiring a specific behavior, rather than
36 tolerating any behavior when using that functionality, is using extensions, as
37 defined in 1.3.2.3. [ISO/IEC 9945-1:1990 (B1)]

1 **2.2.2 General Terms**

2 For the purposes of this standard, the following definitions apply:

3 **2.2.2.1 abstract class:** A class, instances of which are forbidden unless they
4 belong to one of its concrete subclasses.

5 **2.2.2.2 accessible object:** An object for which the client possesses a valid designator or handle.
6

7 **2.2.2.3 assertion:** A statement that is derived from the standard to which conformance is being measured, that is true for a conforming implementation, and
8 that pertains either to functionality or behavior of a functional interface or
9 namespace allocation or to the documentation associated with the implementation
10 being tested.
11

12 **2.2.2.4 attribute:** A component of an object, comprised of an integer denoting the type of the attribute and an ordered sequence of one or more attribute values,
13 each accompanied by an integer denoting the syntax of the value.
14

15 **2.2.2.5 bit string:** An ordered sequence of zero or more bits.

16 **2.2.2.6 byte:** A unit of machine storage containing an ordered sequence of 8 b.

17 **2.2.2.7 character string:** An ordered sequence of zero or more characters.

18 **2.2.2.8 class:** A category into which objects are placed on the basis of both their purpose and their internal structure.
19

20 **2.2.2.9 client:** Software that uses the interface.

21 **2.2.2.10 concrete class:** A class, instances of which are permitted either by direct instantiation or the instantiation of its concrete subclasses.
22

23 **2.2.2.11 datatype:** A collection of distinguished values, together with a collection of characterizing operations on those values.
24

25 **2.2.2.12 element:** Any of the bits of a bit string, the octets of an octet string, or the bytes by means of which the characters of a character string are represented.
26

27 **2.2.2.13 generic interface:** The interface, defined at a level that is independent of any particular programming language.
28

1 **2.2.2.14 immediate subclass:** A subclass, of a class *C*, having no superclasses
2 that are themselves subclasses of *C*.

3 **2.2.2.15 immediate subobject:** One object that is a value of an attribute of
4 another.

5 **2.2.2.16 immediate superclass:** The superclass, of a class *C*, having no sub-
6 classes that are themselves superclasses of *C*.

7 **2.2.2.17 immediate superobject:** One object that contains another among its
8 attribute values.

9 **2.2.2.18 inaccessible:** An object for which the client does not possess a valid
10 designator or handle.

11 **2.2.2.19 independent copy:** A copy of the object plus independent copies of all
12 its subobjects (applied recursively).

13 **2.2.2.20 instance:** An object in the category represented by a class.

14 **2.2.2.21 interface:** An abbreviation for the definition in 2.2.2.30.

15 **2.2.2.22 intermediate datatype:** Any of the basic datatypes in terms of which
16 the other, substantive datatypes of the interface are defined.

17 **2.2.2.23 length:** The number of bytes required to represent any string.

18 **2.2.2.24 minimally consistent object:** An object that satisfies various condi-
19 tions set forth in the definition of its class.

20 **2.2.2.25 name:** An identifier used to represent a specific value of a specific
21 datatype.

22 This standard does not specify when a name is bound to its value: at compile time,
23 link time, or load time. The use of a name as a parameter in a datatype
24 specification does not imply that its value is defined at compile time.

25 **2.2.2.26 object:** Any of the complex information objects created, examined,
26 modified, or destroyed by means of the interface.

27 **2.2.2.27 octet:** An ordered sequence of 8 b.

28 NOTE: Octets can be stored in larger objects if appropriate to a particular architecture.

- 1 **2.2.2.28 octet string:** A string comprised of octets.
- 2 **2.2.2.29 operation:** A program abstraction with formal input and output param-
3 eters that are bound to objects or values in the application that invokes it.
- 4 **2.2.2.30 OSI abstract data manipulation application program interface:**
5 The interface defined by this standard.
- 6 **2.2.2.31 package:** A group of related classes.
- 7 **2.2.2.32 package closure:** The set of classes that need to be supported in order
8 to be able to create all possible instances of all classes defined in the package.
- 9 **2.2.2.33 position (within a string):** The ordinal position of one element of a
10 string relative to another.
- 11 **2.2.2.34 position (within an attribute):** The ordinal position of one value rela-
12 tive to another.
- 13 **2.2.2.35 primary representation:** The form in which the service supplies an
14 attribute value to the client.
- 15 **2.2.2.36 private object:** The internal representation of the language-
16 independent model of an object in a service, and thus unspecified.
- 17 **2.2.2.37 programming language binding specification:** For a language-
18 independent specification, a document that specifies, in terms of a particular pro-
19 gramming language, the behavior that the language-independent specification
20 specifies in language-independent terms.
- 21 It may also specify additional behavior that is relevant to the usage of the particu-
22 lar programming language.
- 23 **2.2.2.38 public object:** The representation by the client of an object in a particu-
24 lar language binding.
- 25 **2.2.2.39 secondary representation:** A second form, an alternative to the pri-
26 mary representation, in which the client may supply an attribute value to the
27 service.
- 28 **2.2.2.40 segment:** Zero or more contiguous elements of a string.
- 29 **2.2.2.41 service:** Software that implements the interface.

- 1 **2.2.2.42 service interface:** The interface as realized, for the benefit of the client,
2 by the service as a whole.
- 3 **2.2.2.43 specific:** With respect to a class, the attribute types that may appear in
4 an instance of the class but not in an instance of its superclasses.
- 5 **2.2.2.44 status datatype:** An abstract datatype whose values may be bound to
6 “control” values as well as “data” values.
- 7 **2.2.2.45 string:** An ordered sequence of zero or more bits, octets, or characters,
8 accompanied by the length of the string.
- 9 **2.2.2.46 subclass:** One of the classes, designated as such, whose attribute types
10 are a superset of those of another class.
- 11 **2.2.2.47 subobject:** An immediate subobject of an object or of one of its subob-
12 jects applied recursively.
- 13 **2.2.2.48 superclass:** One of the classes, designated as such, whose attribute
14 types are a subset of those of another class.
- 15 **2.2.2.49 superobject:** The immediate superobject of an object, or one of its
16 superobjects applied recursively.
- 17 **2.2.2.50 syntax:** A category into which an attribute value is placed on the basis
18 of its form.
- 19 **2.2.2.51 syntax template:** A lexical construct containing an asterisk from which
20 several attribute syntaxes can be derived by substituting text for the asterisk.
- 21 **2.2.2.52 test methods:** The software, procedures, or other means specified to
22 measure conformance to a specification.
- 23 **2.2.2.53 type:** A category into which attribute values are placed on the basis of
24 their purpose.
- 25 **2.2.2.54 value:** An arbitrarily complex information item that can be viewed as a
26 characteristic or property of an object.
- 27 **2.2.2.55 workspace:** A repository for instances of classes in the closures of one
28 or more packages associated with the workspace.

1 **2.2.3 Abbreviations**

2 For the purposes of this standard, the following abbreviations apply:

3 **2.2.3.1 API:** Application Program Interface.

4 **2.2.3.2 ASN.1:** Abstract Syntax Notation One (defined in ISO/IEC 8824:1990 {1}).

5 **2.2.3.3 BER:** Basic Encoding Rules (defined in ISO/IEC 8825:1990 {2}).

6 **2.2.3.4 IA5:** International Alphabet No. 5 (used in “IA5 String”; see ISO/IEC
7 8824:1990 {1}).

8 **2.2.3.5 OM:** OSI abstract data Manipulation.

9 **2.2.3.6 OSI:** Open Systems Interconnection.

10 **2.2.3.7 UTC:** Universal Coordinated Time.

1

Section 3: Overview

2 This section introduces the interface and its specification. It indicates the purpose
3 of the interface, provides the motivation for it, identifies the levels of abstraction
4 at which the interface is defined, and explains how identifiers at one level are
5 derived from those at the other.

6 3.1 Purpose

7 This document defines a general-purpose OSI Object Management Application
8 Program Interface (API) for use in conjunction with, but otherwise independent of,
9 other application-specific APIs for Open Systems Interconnection (OSI).

10 Object Management (OM) is the creation, examination, modification, and deletion
11 of potentially complex information objects.⁵⁾ It presents to programmers a uniform
12 model, or architecture, of information based upon the concept of groups, or classes,
13 of similar information objects. The OM API provides facilities to manipulate both
14 small objects and those too large to be held in main memory.

15 The information objects to which OM applies are those that arise in OSI, i.e.,
16 those that correspond to the types defined by, or by means of, Abstract Syntax
17 Notation One (ASN.1). The OM API comprises tools for manipulating ASN.1
18 objects. It shields the programmer from much of the complexity of ASN.1, e.g., its
19 Basic Encoding Rules (BER).

20 This API is designed to work with groups of ASN.1 objects that are called
21 packages. These packages are defined by the application-specific APIs that use
22 this API. The packages contain all the ASN.1 objects necessary to accomplish a
23 specific task. Thus, the API does not work with any arbitrary ASN.1 objects, only
24 those defined in packages.

25 The OM API is designed to be implemented by one or more manufacturers work-
26 ing independently. As illustrated in Figure 3-1 below, each manufacturer effec-
27 tively provides the programmer with the ability to manipulate information objects

28

29 5) These objects, which are described in Section 4 of this standard, differ from the directory objects defined in ISO
30 9594-1: 1990 {B2}.

of particular kinds. This division of implementation responsibility is achieved by means of workspaces (see 4.8).

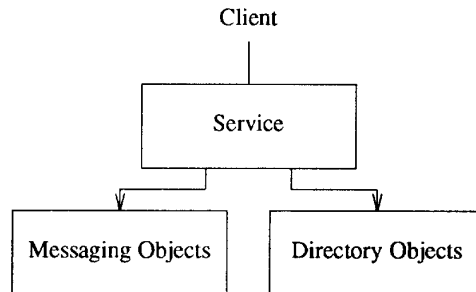


Figure 3-1 – Conceptual Model of Object Management

Throughout this document, the term *interface* denotes the OM API, the term *service* denotes software that supplies (i.e., implements) the interface, and the term *client* denotes software that uses the interface. The term *service interface* denotes the interface realized by the service as a whole, and it is thus a synonym for *interface*.

3.2 Motivation

The OM interface is designed to be used with other, application-specific APIs, e.g., for message transfer or directory access. Such an API typically defines both a set of operations and a set of structured information objects that serve as the arguments and results of the operation. The OM interface defines a general information architecture for structuring such information objects, as well as general operations for manipulating them. Both the architecture and the operations are independent of the application-specific APIs they support.

The OM information architecture is object-oriented and thus enjoys the modularity and extensibility inherent in that approach. Collections of data are referred to as objects, objects of like structure are grouped together into classes, and classes are related to one another by means of subclassing. The information architecture is described in detail in Section 4.

Among its important features, however, is that wherever an application-specific API requires an instance of a particular class, an instance of any of the subclasses of that class may be supplied instead. This permits the refinement and extension of objects and the use of those extended objects in application-specific APIs.

NOTE: While OM takes an object-oriented view of structured information, it does not incorporate all the characteristics of other object-oriented systems. In particular, the definitions of the

1 operations for manipulating objects are separate from the definitions of the classes of the objects,
2 and there is no notion of encapsulating or hiding the information associated with objects (although
3 the interface hides the representation of the information).

4 The OM interface is specifically designed for use with application-specific APIs
5 that provide OSI services. The objects addressed by the information architecture,
6 therefore, are those arising from ASN.1, the descriptive tool used pervasively in
7 OSI. By providing tools for manipulating ASN.1 objects, the OM interface shields
8 the client from much (but not all) of the complexity of ASN.1.

9 While it presents to the client a single model of information, the OM interface nei-
10 ther defines nor unnecessarily constrains the representation of the objects that the
11 service maintains internally. The interface is designed as a general information
12 management facility that can accommodate the varied objects required by
13 application-specific OSI APIs. Therefore, no constraints are placed on the struc-
14 ture, size, or location of the information objects held by the service. The internal
15 representation of such objects, which may be application-specific, is hidden from
16 the client; the objects can be accessed only using OM interface operations.

17 Objects are conveyed between the client and the service, in whole or in part, using
18 language-independent constructs that are described in the generic interface
19 definition. The representation of these constructs shall be specified in each
20 language-dependent realization of the generic interface.

21 The extent to which the OM interface is able to hide the internal representation of
22 objects is insufficient to meet fully the needs of environments supporting several
23 application-specific APIs. Such APIs may impose varied and even conflicting
24 requirements upon the internal representations of objects, and they may even be
25 implemented by different vendors. Therefore, the OM interface is designed to per-
26 mit any number of OM interface implementations to coexist, each representing
27 objects differently. This is accomplished by means of workspaces (see 4.8).

28 The various OM interface implementations cannot be completely independent.
29 Two different application-specific services using two different OM interface imple-
30 mentations may have to exchange information. A message transfer system, for
31 example, may require a name or address obtained from a directory system. The
32 OM interface enables such information exchange by providing a single OM inter-
33 face (used by all application-specific services) to any number of implementations of
34 that interface (workspaces). While it recognizes that different workspaces handle
35 objects of different kinds, the client need not explicitly move information from
36 workspace to workspace in order to convey it effectively from one application-
37 specific service to another.

38 The mechanism for communication between the workspaces of different service
39 providers is language dependent and may not be possible in all programming
40 languages. The concept is left intentionally vague in the language-independent
41 definition to facilitate future language bindings. However, it is the intent of the
42 language-independent specification that a single invocation of an operation by the
43 client will result in the correct workspace being used by the service.