

INTERNATIONAL STANDARD

ISO/IEC
14360

First edition
1996-06-01

Information technology — Open Systems Interconnection (OSI) abstract data manipulation — Application Program Interface (API) [Language independent]

*Technologies de l'information — Manipulation de données abstraites en
interconnexion de systèmes ouverts (OSI) — Interface de programme
d'application (API) [indépendante du langage]*



Reference number
ISO/IEC 14360-1996(E)

AVA Documents
Document Sharing Hub

Contents

	PAGE
Section 1: General	1
1.1 Scope	1
1.2 Normative References	2
1.3 Conformance	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	22
4.5 Class Definitions	24
4.6 Packages	25
4.7 Package Definitions	25
4.8 Workspaces	25
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	31
5.7 Relationship to ASN.1 Character String Types	32
5.8 Relationship to ASN.1 Type Constructors	33

© ISO/IEC 1996

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Section 6: Service Interface	35
6.1 Introduction	35
6.2 Datatypes	35
6.2.1 Signed Integers	35
6.2.2 Unsigned Integers	36
6.2.3 Boolean	36
6.2.4 Enumeration	37
6.2.5 Exclusions	37
6.2.6 Integer	38
6.2.7 Modification	38
6.2.8 Object	38
6.2.9 Object Identifier	39
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 Real	44
6.2.23 Return Code	44
6.2.24 String	45
6.2.25 Type	45
6.2.26 Type List	46
6.2.27 Value Length	46
6.2.28 Value Position	46
6.2.29 Workspace	47
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	61
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	73
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	32
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	71
Table 7-2 – Attributes Specific to External	72
Table 7-3 – Attributes Specific to Object	73
Table 8-1 – Class Object Identifier Constant Suffixes	75
Table 8-2 – Attribute Type Identifiers	76

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 14360 was prepared by IEEE (as IEEE Std 1224-1993) 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 14360, Information technology—Open Systems Interconnection (OSI) abstract data manipulation—Application Program Interface (API) [Language independent], but is included for information only.)

The purpose of this International Standard is to define a general-purpose OSI Abstract Data Manipulation (OM) Application Program 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.

11 Related Standards

This International 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 ISO/IEC 14364 {B4}.

This International 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 ISO/IEC 14361 {B3} and the API to directory services defined in ISO/IEC 14392 {B6}.

ISO/IEC 14362 {4} contains a set of requirements to be satisfied by test methods for measuring conformance to this International 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 International Standard.

25 Overview

This International 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.

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

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

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

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

50 Related Standards Activities

51 The following areas are under active consideration at this time, or are expected to
52 become active in the near future, concerning standards for application APIs that
53 use the mechanism defined in this International Standard. Similar efforts can be
54 anticipated in the future:¹⁾

- 55 (1) X.400-based message handling
- 56 (2) Directory services
- 57 (3) FTAM API
- 58 (4) Verification testing methods
- 59 (5) Network interface facilities
- 60 (6) System administration.

This International Standard is based on IEEE Std 1224-1993 {B7}, which was prepared by the P1224 Working Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society.

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

Information technology—Open Systems Interconnection (OSI) abstract data manipulation—Application Program Interface (API) [Language independent]

Section 1: General

1.1 Scope

This International Standard defines a standard interface supporting the manipulation of complex arguments and parameters used by X.400 and Directory Services APIs. The interface supports manipulation of abstract data defined in ASN.1 and is for use in conjunction with, but is otherwise independent of, the X.400 and Directory Services APIs.

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

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

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

1.2 Normative References

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

- [1] ISO/IEC 8824: 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] 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]*.

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

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

- (1) The implementation shall support all required behavior defined in this International 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 International Standard and in the programming language specification.

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

2) CCITT documents can be obtained from the Telecommunication Standardization Bureau of the International Telecommunication Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

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

1.3.1.2 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to a programming language binding specification for this International Standard. The conformance document shall be in two parts. The first part shall have the same structure as this International Standard, with the information presented in the appropriately numbered sections, clauses, and subclauses. The second part shall have the same structure as the programming language binding specification, with the information presented in the appropriately numbered sections, clauses, and subclauses. The conformance document shall not contain information about extended features or capabilities outside the scope of this International Standard and the programming language binding specification.

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

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

The conformance document shall state which of the optional features defined in this International Standard and in the programming language binding specification are supported by the implementation.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in this International Standard and in the programming language binding specification. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. The conformance document may specify the behavior of the implementation for those

3) The numbers in curly brackets correspond to those of the references in 1.2.

features where this International 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 International Standard and the programming language binding specification shall be present in the conformance document.

The phrases “shall document” or “shall be documented” in this International Standard or in a programming language binding specification for this International 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 International 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 International Standard, in the programming language binding specification, and in the applicable language standards. Such an application shall accept any behavior described in this International 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 International Standard and the programming language binding specification.

1.3.2.2 Conforming Application

There is only one type of Conforming Application.

1.3.2.2.1 ISO/IEC Conforming Application

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

1.3.2.3 Conforming Application Using Extensions

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

1.4 Test Methods

Any measurement of conformance to a programming language binding specification for this International Standard shall be performed using test methods that conform to ISO/IEC 14362 {4} and to any additional requirements that may be imposed by the programming language binding specification.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 General and Typographical Conventions

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

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

The use of fonts in this International Standard is as follows.

- The Helvetica font is used for:

- Language-independent operation names, such as `om_copy_value`

- Language-independent datatype names, such as `om_exclusions_type`

- Language-independent error names, such as `no_such_class`

- The *italic* font is used for:

- Language-independent operation arguments, such as *source_value_position*

- The introduction of important terms

- Cross-references in 2.2

- The **bold** font is used for:

- Language-independent concrete OM class names, such as **External**

- Language-independent OM attribute names, such as **Arbitrary-Encoding**

- Language-independent OM attribute values, such as **insert-at-certain-point**

- The ***bold italic*** font is used for:

- Language-independent abstract OM class names, such as ***Object***

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

- References to terms defined in ISO/IEC 8824 {1}, ISO/IEC 8825 {2}, and ISO/IEC 9594-8 {3}.

2.1.2 Language-Independent Conventions

NOTE: In the absence of a normative notation for language-independent specifications, these conventions are based upon a draft of the IEEE P1003.1LIS conventions {B9}.

This subclause defines the notation used within this International Standard to specify the information that is communicated through interfaces to the defined services. The notation assumes a model of an abstract procedural interface, although language bindings to this International Standard may take other forms.

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

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

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

2.1.2.1 Notation

A modified backus-naur form (BNF) notation is used in this subclause. The characters “:”, “=”, “<”, “>”, “[”, “{”, and “)” are always part of this notation and are never used in terminal strings.

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

2.1.2.2 Identifiers

```

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

```

2.1.2.3 Numbers

```

70 <number> ::= <digit> | <digit> <number>
71 <signed-number> ::= { - } <number>

```

2.1.2.4 Datatypes

The following notation is used to specify a datatype:

```

74 Datatype: <datatype-name>
75 Definition: <datatype>
76 Description: ...

```

where

```

78 <datatype> ::= <datatype-name> | <datatype-definition>
79 <datatype-name> ::= <identifier>
80 <datatype-definition> ::= <atomic-type> | <constructed-type>
81 <atomic-type> ::= boolean_type | <opaque-type> | <state-type>
82 <ordinal-type> | <integer-type> | <real-type>
83 <opaque-type> ::= opaque [ <size> ]
84 <state-type> ::= state [ <state-list> ]
85 <state-list> ::= <state-value> { , <state-list> }
86 <state-value> ::= <identifier>
87 <ordinal-type> ::= ordinal_type [ <size> ]
88 <size> ::= <number>
89 <integer-type> ::= integer_type [ <minimum> , <maximum> ]
90 <minimum> ::= <signed-number>
91 <maximum> ::= <signed-number>
92 <constructed-type> ::= <array-type> | <record-type> |
93 <set-type> | <choice-type> | <set-type>
94 <array-type> ::= array <index-type> of <base-type>
95 <index-type> ::= <ordinal-type> | <integer-type> | <state-type>
96 <base-type> ::= <datatype>
97 <record-type> ::= record [ <member-list> ]
98 <choice-type> ::= choice [ <member-list> ]
99 <set-type> ::= set [ <datatype> ]
100 <list-type> ::= list [ <datatype> ]
101 <member-list> ::= <member-definition> { , <member-list> }

```

<member-definition> ::= <member-name> : <member-type>

<member-name> ::= <identifier>

<member-type> ::= <datatype>

Any of the values within square brackets may be listed as *<implementation defined>*; see 2.2.1.2 for the requirements imposed by this notation.

2.1.2.5 Names

The following notation is used to specify a name:

Name:	<identifier>
Datatype:	<datatype-definition>
Description:	...

2.1.2.6 Operations

The following notation is used to specify a operation:

Operation:	<identifier>
Status Datatype:	<datatype>
Input:	<parameter-name> : <datatype>
	<parameter-name> : <datatype>
	...
Output:	<parameter-name> : <datatype>
	<parameter-name> : <datatype>
	...

where

<parameter-name> ::= <identifier>

2.2 Definitions

2.2.1 Terminology

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

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

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

2.2.1.3 may: An indication of an optional feature.

With respect to implementations, the word *may* is to be interpreted as an optional feature that is not required in this International Standard, but that can be provided. With respect to Strictly Conforming Applications, the word *may* means that the optional feature shall not be used. [ISO/IEC 9945-1 {B2}]

2.2.1.4 shall: An indication of a requirement on the implementation or on Strictly Conforming Applications, where appropriate. [ISO/IEC 9945-1 {B2}]**2.2.1.5 should:**

- (1) With respect to implementations, an indication of an implementation recommendation, but not a requirement.
- (2) With respect to applications, an indication of a recommended programming practice for applications and a requirement for Strictly Conforming Applications. [ISO/IEC 9945-1 {B2}]

2.2.1.6 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 by 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 {B2}]

2.2.1.7 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 {B2}]

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

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

2.2.2 General Terms

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

168 **2.2.2.1 abstract class:** A class, instances of which are forbidden unless they
169 belong to one of its concrete subclasses.

170 **2.2.2.2 accessible object:** An object for which the client possesses a valid desig-
171 nator or handle.

172 **2.2.2.3 assertion:** A statement that is derived from the standard to which confor-
173 mance is being measured, that is true for a conforming implementation, and that
174 pertains either to functionality or behavior of a functional interface or namespace
175 allocation or to the documentation associated with the implementation being
176 tested.

177 **2.2.2.4 attribute:** A component of an object, comprised of an integer denoting the
178 type of the attribute and an ordered sequence of one or more attribute values, each
179 accompanied by an integer denoting the syntax of the value.

180 **2.2.2.5 bit string:** An ordered sequence of zero or more bits.

181 **2.2.2.6 byte:** A unit of machine storage containing an ordered sequence of 8 b.

182 **2.2.2.7 character string:** An ordered sequence of zero or more characters.

183 **2.2.2.8 class:** A category into which objects are placed on the basis of both their
184 purpose and their internal structure.

185 **2.2.2.9 client:** Software that uses the interface.

186 **2.2.2.10 concrete class:** A class, instances of which are permitted either by
187 direct instantiation or the instantiation of its concrete subclasses.

188 **2.2.2.11 datatype:** A collection of distinguished values, together with a collection
189 of characterizing operations on those values.

190 **2.2.2.12 element:** Any of the bits of a bit string, the octets of an octet string, or
191 the bytes by means of which the characters of a character string are represented.

192 **2.2.2.13 generic interface:** The interface, defined at a level that is independent
193 of any particular programming language.

194 **2.2.2.14 immediate subclass:** A subclass, of a class *C*, having no superclasses
195 that are themselves subclasses of *C*.

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

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

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

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

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

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

2.2.2.21 interface: An abbreviation for the definition in 2.2.2.30.

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

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

2.2.2.24 minimally consistent object: An object that satisfies various conditions set forth in the definition of its class.

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

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

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

2.2.2.27 octet: An ordered sequence of 8 b.

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

2.2.2.28 octet string: A string comprised of octets.

2.2.2.29 operation: A program abstraction with formal input and output parameters that are bound to objects or values in the application that invokes it.

2.2.2.30 OSI abstract data manipulation application program interface:
The interface defined by this International Standard.

2.2.2.31 package: A group of related classes.

2.2.2.32 package closure: The set of classes that need to be supported in order to be able to create all possible instances of all classes defined in the package.

2.2.2.33 position (within a string): The ordinal position of one element of a string relative to another.

2.2.2.34 position (within an attribute): The ordinal position of one value relative to another.

2.2.2.35 primary representation: The form in which the service supplies an attribute value to the client.

2.2.2.36 private object: The internal representation of the language-independent model of an object in a service, and thus unspecified.

2.2.2.37 programming language binding specification: For a language-independent specification, a document that specifies, in terms of a particular programming language, the behavior that the language-independent specification specifies in language-independent terms.

It may also specify additional behavior that is relevant to the usage of the particular programming language.

2.2.2.38 public object: The representation by the client of an object in a particular language binding.

2.2.2.39 secondary representation: A second form, an alternative to the primary representation, in which the client may supply an attribute value to the service.

2.2.2.40 segment: Zero or more contiguous elements of a string.

2.2.2.41 service: Software that implements the interface.

2.2.2.42 service interface: The interface as realized, for the benefit of the client, by the service as a whole.

2.2.2.43 specific: With respect to a class, the attribute types that may appear in an instance of the class but not in an instance of its superclasses.

- 253 **2.2.2.44 status datatype:** An abstract datatype whose values may be bound to
“control” values as well as “data” values.
- 254 **2.2.2.45 string:** An ordered sequence of zero or more bits, octets, or characters,
255 accompanied by the length of the string.
- 256 **2.2.2.46 subclass:** One of the classes, designated as such, whose attribute types
257 are a superset of those of another class.
- 258 **2.2.2.47 subobject:** An immediate subobject of an object or of one of its subobjects
259 applied recursively.
- 260 **2.2.2.48 superclass:** One of the classes, designated as such, whose attribute
261 types are a subset of those of another class.
- 262 **2.2.2.49 superobject:** The immediate superobject of an object, or one of its
263 superobjects applied recursively.
- 264 **2.2.2.50 syntax:** A category into which an attribute value is placed on the basis of
265 its form.
- 266 **2.2.2.51 syntax template:** A lexical construct containing an asterisk from which
267 several attribute syntaxes can be derived by substituting text for the asterisk.
- 268 **2.2.2.52 test methods:** The software, procedures, or other means specified to
269 measure conformance to a specification.
- 270 **2.2.2.53 type:** A category into which attribute values are placed on the basis of
271 their purpose.
- 272 **2.2.2.54 value:** An arbitrarily complex information item that can be viewed as a
273 characteristic or property of an object.
- 274 **2.2.2.55 workspace:** A repository for instances of classes in the closures of one or
275 more packages associated with the workspace.

2.2.3 Abbreviations

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

277 **2.2.3.1 API:** Application Program Interface.

278 **2.2.3.2 ASN.1:** Abstract Syntax Notation One (defined in ISO/IEC 8824 {1}).

279 **2.2.3.3 BER:** Basic Encoding Rules (defined in ISO/IEC 8825 {2}).

280 **2.2.3.4 IA5:** International Alphabet No. 5 (used in “IA5 String”; see ISO/IEC 8824
281 {1}).

282 **2.2.3.5 OM:** OSI abstract data Manipulation.

283 **2.2.3.6 OSI:** Open Systems Interconnection.

284 **2.2.3.7 UTC:** Universal Coordinated Time.

Section 3: Overview

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

3.1 Purpose

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

Object Management (OM) is the creation, examination, modification, and deletion of potentially complex information objects.⁴⁾ It presents to programmers a uniform model, or architecture, of information based upon the concept of groups, or classes, of similar information objects. The OM API provides facilities to manipulate both small objects and those too large to be held in main memory.

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

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

The OM API is designed to be implemented by one or more manufacturers working independently. As illustrated in Figure 3-1 below, each manufacturer effectively provides the programmer with the ability to manipulate information objects of par-

⁴⁾ These objects, which are described in Section 4 of this International Standard, differ from the directory objects defined in ISO 9594-1 (B1).

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

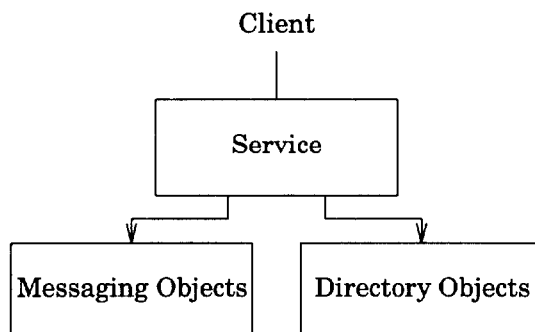


Figure 3-1 – Conceptual Model of Object Management

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

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

While it presents to the client a single model of information, the OM interface neither defines nor unnecessarily constrains the representation of the objects that the service maintains internally. The interface is designed as a general information management facility that can accommodate the varied objects required by application-specific OSI APIs. Therefore, no constraints are placed on the structure, size, or location of the information objects held by the service. The internal representation of such objects, which may be application-specific, is hidden from the client; the objects can be accessed only using OM interface operations.

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

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

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

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

Section 4: Information Architecture

4.1 Introduction

This section specifies the architecture of the information that the client and service exchange, and that the service maintains and makes accessible to the client.

The architecture provides a basis for specifying how the client communicates with the service, how the service communicates with the client in response, and how components of the service communicate with one another. The purpose of the architecture is not to dictate the physical structure of information as the service maintains it internally; this is implementation-specific and thus unspecified.

4.2 Objects

The principal purpose of the service is to create, examine, modify and destroy complex information objects under the direction of the client. A principal purpose of the interface is to enable the client and the service to exchange objects in whole or in part. This requirement provides the rationale for the information architecture whose specification follows.

Objects are of two kinds: public and private. A “public object” is represented by a data structure whose format is part of the service’s specification (see 6.2). Therefore the client creates, examines, modifies and destroys public objects directly, i.e., by means of programming language constructs. The representation of a private object is unspecified. The client therefore accesses private objects only indirectly, i.e., by means of interface operations.

NOTE: Public objects simplify application programs, enabling them (where appropriate) to define objects statically, rather than requiring them to construct the objects dynamically by means of sequences of interface operation invocations.

The interface is comprised of operations for both examining and modifying private objects. For application-specific reasons, however, the service may deny a request from the client to modify a particular object at a particular time. The specification of each application-specific API shall identify any circumstances under which this may occur.

Public objects themselves are of two kinds, client- and service-generated. A “client-generated” public object is fabricated by the client in storage it provides. A “service-generated” public object is fabricated by the service in storage it provides.