
**Information technology — Open Systems
Interconnection (OSI) abstract data
manipulation C language interfaces —
Binding for Application Program Interface
(API)**

*Technologies de l'information — Interfaces de langage C pour la
manipulation de données abstraites en interconnexion de systèmes
ouverts (OSI) — Liant pour l'interface de programme d'application (API)*



Reference number
ISO/IEC 14364:1996(E)

Contents

	PAGE
Section 1: General	1
1.1 Scope	1
1.2 Normative References	1
1.3 Conformance	2
1.4 Test Methods	3
Section 2: Terminology and General Requirements	5
2.1 Conventions	5
2.2 Definitions	7
Section 3: Service Interface	15
3.1 Introduction	15
3.2 Datatypes	15
3.2.1 Intermediate Datatypes	15
3.2.2 Boolean	16
3.2.3 Descriptor	16
3.2.4 Enumeration	18
3.2.5 Exclusions	18
3.2.6 Integer	19
3.2.7 Modification	19
3.2.8 Object	20
3.2.9 Object Identifier	20
3.2.10 Private Object	22
3.2.11 Public Object	22
3.2.12 Real	24
3.2.13 Return Code	24
3.2.14 String	25
3.2.15 String Length	26
3.2.16 Syntax	27
3.2.17 Type	28
3.2.18 Type List	28
3.2.19 Value	29
3.2.20 Value Length	30
3.2.21 Value Position	31
3.2.22 Workspace	31

© 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

3.3 Functions	32
3.3.1 Copy	32
3.3.2 Copy Value	32
3.3.3 Create	33
3.3.4 Decode	33
3.3.5 Delete	33
3.3.6 Encode	34
3.3.7 Get	34
3.3.8 Instance	35
3.3.9 Put	35
3.3.10 Read	36
3.3.11 Remove	37
3.3.12 Write	37
3.4 Return Codes	38
3.5 Header File	38
Section 4: Workspace Interface	39
4.1 Introduction	39
4.2 Representation of Objects	39
4.3 Types and Macros	41
4.3.1 Standard Internal Representation of an Object	41
4.3.2 Standard Internal Representation of a Workspace	41
4.3.3 Useful Macros	44
4.4 Dispatcher Macros	45
Section 5: Definitions of Constants	49
5.1 Exclusions	49
5.2 Modification	49
5.3 Object Identifiers	50
5.4 Return Code	50
5.5 String (Elements Component)	50
5.6 Syntax	50
5.7 Type	50
Annex A (informative) Example <i><xom.h></i> Header	53
Annex B (informative) Bibliography	61
Alphabetic Topical Index	63

FIGURES

Figure 3-1 – Example Descriptor Initialization	23
Figure 3-2 – Bit String Representation	26

TABLES

Table 2-1 – Naming Conventions 6

Table 3-1 – Symbolic Names for Object Identifiers 21

Table 5-1 – Exclusions Constants 49

Table 5-2 – Modification Constants 49

Table 5-3 – Object Identifier Constants 50

Table 5-4 – Return Code Constants 51

Table 5-5 – Elements Unspecified Constant 51

Table 5-6 – Syntax Constants 52

Table 5-7 – Type Constants 52

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 14364 was prepared by IEEE (as IEEE Std 1327-1993) 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.

Annexes A and B of this International Standard are for information only.

1 Introduction

(This introduction is not a normative part of ISO/IEC 14364, Information technology—Open Systems Interconnection (OSI) abstract data manipulation C language interfaces—Binding for Application Program Interface (API), but is included for information only.)

The purpose of this International Standard is to define a C programming language binding for the general-purpose OSI Abstract Data Manipulation (OM) Application Program Interface (API) that is defined in ISO/IEC 14360 {2}.

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

The programming language independent specification on which the programming language binding defined in this International Standard is based is contained in ISO/IEC 14360 {2}.

This International Standard is intended to be used in conjunction with C programming language bindings of application-specific APIs that provide OSI services, such as the X.400-Based Messaging API defined in ISO/IEC 14365 {B5} and the API to Directory Services defined in ISO/IEC 14394 {B6}.

ISO/IEC 14366 {4} specifies a set of requirements that need to be satisfied for measuring conformance to this International Standard. In addition, test methods must also satisfy the requirements specified in ISO/IEC 14362 {3} for measuring conformance to the programming language-independent specification of the API.

23 Overview

The API defined in this International Standard contains two major parts. One consists of the interface for accessing the services defined in the programming language-independent specification. It is defined in Section 3. The other consists of an interface for language-specific services. It is defined in Section 4. Any implementation claiming conformance to this International Standard must provide both parts of the API.

Related Standards Activities

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

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

This International Standard is based on IEEE Std 1327-1993 {B8}, 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 C language interfaces—Binding for Application Program Interface (API)

Section 1: General

1.1 Scope

This International Standard defines an ISO/IEC 9989 {1}¹⁾ C language binding to the language-independent API defined in ISO/IEC 14360 {2}. It is intended to support application portability at the source-code level and to be used by application developers, system implementors, test method writers, and users.

1.2 Normative References

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

{1} ISO/IEC 9899:1990,²⁾ *Programming Languages—C*.

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

2) 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} ISO/IEC 14360: 1996, *Information technology—Open Systems Interconnection (OSI) abstract data manipulation—Application Program Interface (API) [Language independent]*.
- {3} ISO/IEC 14362: 1996, *Information technology—Test methods for measuring conformance to Open Systems Interconnection (OSI) abstract data manipulation—Application Program Interface (API) [Language independent]*.
- {4} ISO/IEC 14366: 1996, *Information technology—Test methods for measuring conformance to Open Systems Interconnection (OSI) abstract data manipulation C language interfaces—Binding for Application Program Interface (API)*.

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

A conforming implementation for this International Standard shall be a conforming implementation, as defined in ISO/IEC 14360 {2}, for the programming language binding specification that is contained in this International Standard.

1.3.1.2 Documentation

A conformance document shall be available for an implementation claiming conformance to this International Standard. It shall satisfy the requirements that are defined in ISO/IEC 14360 {2} for a conformance document for a programming language binding specification.

The phrases “shall document” or “shall be documented” in this International Standard mean that documentation of the feature shall appear in the conformance document, 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

In addition to those aspects described in ISO/IEC 14360 {2}, the following aspect of the behavior of the service is implementation defined:

- the precise definitions in C of the intermediate datatypes.

1.3.2 Application Conformance

All applications claiming conformance to this International Standard shall fall within one of the categories in the following subclauses.

1.3.2.1 Strictly Conforming Application

A Strictly Conforming Application for this International Standard is a Strictly Conforming Application as defined in ISO/IEC 14360 {2} for the programming language binding specification contained in this International Standard.

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 for this International Standard is an ISO/IEC Conforming Application as defined in ISO/IEC 14360 {2} for the programming language binding specification contained in this International Standard.

1.3.2.3 Conforming Application Using Extensions

A Conforming Application Using Extensions for this International Standard is a Conforming Application Using Extensions as defined in ISO/IEC 14360 {2} for the programming language binding specification contained in this International Standard.

1.4 Test Methods

Any measurement of conformance to this International Standard shall be performed using test methods that conform to ISO/IEC 14362 {3} and to ISO/IEC 14366 {4}.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 General and Typographic Conventions

When presented in **Synopsis** sections or in extended code fragments, C language source code is printed using `fixed width font`. Where C language names or code fragments appear in the text of this International Standard, the following conventions apply:

- A C language function, datatype, or code fragment appears in *italic font*. A function name is indicated by following parentheses, for example, *om_get()*.
- A symbolic constant, other than an error name, is surrounded by braces, for example, {OM_ARBITRARY_ENCODING}.
- A symbolic constant for an error name is surrounded by brackets, for example, [OM_FUNCTION_DECLINED].

2.1.2 C Naming Conventions

The C language binding specifies C identifiers for all the elements of the interface. These elements include function names, typedef names, and constants. All the C identifiers are mechanically derived from the language-independent names.

The interface uses part of the C public namespace for its facilities. All identifiers that start with the letters *om*, *omP*, *OM*, or *OMP* are reserved. It also reserves all identifiers starting with the letters *omX* or *OMX* for extensions of the interface. A list of the conventions used is given in Table 2-1.

All identifiers used (except those beginning with *omP*, *omX*, *OMX*, or *OMP*) are defined in this International Standard. No implementation of the interface shall use any other public identifiers. A *public identifier* is any name except those reserved in the C Standard (ISO/IEC 9899 {1}), and the *public namespace* is the set of all possible public identifiers.

Table 2-1 – Naming Conventions

Item	Prefix
reserved for implementors	omP
reserved for interface extensions	omX
reserved for implementors	OMP
reserved for interface extensions	OMX
functions	om_
OM class names	OM_C_
other constants	OM_

The process for deriving C names from those of the language-independent specification is as follows:

- The C function names are identical to the language-independent operation names.

Thus

`om_copy_val`

becomes

`om_copy_val()`

- C function parameters are identical to the corresponding language-independent argument names.

included_types

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

Thus

Object

becomes

{OM_C_OBJECT}

- For enumeration constants, OM attributes, errors, and all other constants, lowercase is converted to uppercase, hyphens are converted to underscores, and the prefix “OM_” is added.

Thus

Arbitrary-Encoding

becomes

{OM_ARBITRARY_ENCODING}

- The sequence of octets for each object identifier is also assigned an identifier for internal use by certain OM macros. These identifiers are all uppercase and are prefixed by “OMP_O_”. See ISO/IEC 14360 {2} for further details on the use of object identifiers.

2.1.3 Language-Independent Conventions

The language-independent conventions defined in ISO/IEC 14360 {2} shall apply in this International Standard.

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 {B4}]

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 {B4}]

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 {B4}]

2.2.1.4 shall: An indication of a requirement on the implementation or on Strictly Conforming Applications, where appropriate. [ISO/IEC 9945-1 {B4}]

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 {B4}]

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 {B4}]

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 {B4}]

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 {B4}]

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 {B4}]

2.2.2 General Terms

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

2.2.2.1 abstract class: A class, instances of which are forbidden unless they belong to one of its concrete subclasses. [ISO/IEC 14360 {2}]

2.2.2.2 accessible: An object for which the client possesses a valid designator or handle [ISO/IEC 14360 {2}]

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 that pertains either to functionality or behavior of a functional interface or namespace allocation or to the documentation associated with the implementation being tested. [ISO/IEC 14360 {2}]

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, each accompanied by an integer denoting the syntax of the value. [ISO/IEC 14360 {2}]

2.2.2.5 bit string: An ordered sequence of zero or more bits. [ISO/IEC 14360 {2}]

2.2.2.6 byte: A unit of machine storage containing an ordered sequence of 8 b. [ISO/IEC 14360 {2}]

2.2.2.7 C interface: The C language binding, defined in terms of ISO/IEC 9899 {1}.

2.2.2.8 character string: An ordered sequence of zero or more characters.
[ISO/IEC 14360 {2}]

2.2.2.9 class: A category into which objects are placed on the basis of both their purpose and their internal structure. [ISO/IEC 14360 {2}]

2.2.2.10 client: Software that uses the interface. [ISO/IEC 14360 {2}]

2.2.2.11 concrete class: A class, instances of which are permitted either by direct instantiation or the instantiation of its concrete subclasses. [ISO/IEC 14360 {2}]

2.2.2.12 datatype: A collection of distinguished values, together with a collection of characterizing operations on those values. [ISO/IEC 14360 {2}]

2.2.2.13 descriptor: The means by which the client and service exchange an attribute value and the integers that denote its representation, type, and syntax.

2.2.2.14 dispatcher: The software that implements the service interface functions using workspace interface functions.

2.2.2.15 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. [ISO/IEC 14360 {2}]

2.2.2.16 generic interface: The interface, defined at a level that is independent of any particular programming language. [ISO/IEC 14360 {2}]

2.2.2.17 immediate subclass: A subclass, of a class *C*, having no superclasses that are themselves subclasses of *C*. [ISO/IEC 14360 {2}]

2.2.2.18 immediate subobject: One object that is a value of an attribute of another. [ISO/IEC 14360 {2}]

2.2.2.19 immediate superclass: The superclass, of a class *C*, having no subclasses that are themselves superclasses of *C*. [ISO/IEC 14360 {2}]

2.2.2.20 immediate superobject: One object that contains another among its attribute values. [ISO/IEC 14360 {2}]

156 **2.2.2.21 inaccessible:** An object for which the client does not possess a valid designator or handle. [ISO/IEC 14360 {2}]

157 **2.2.2.22 independent copy:** A copy of the object plus independent copies of all
158 its subobjects (applied recursively). [ISO/IEC 14360 {2}]

159 **2.2.2.23 instance:** An object in the category represented by a class. [ISO/IEC
160 14360 {2}]

161 **2.2.2.24 interface:** An abbreviation for the definition in 2.2.2.34.

162 **2.2.2.25 intermediate datatype:** Any of the basic datatypes in terms of which
163 the other, substantive datatypes of the interface are defined. [ISO/IEC 14360 {2}]

164 **2.2.2.26 intermediate macro:** Any of the basic macros in terms of which the
165 other, substantive macros used to realize the dispatcher are defined.

166 **2.2.2.27 length:** The number of bytes required to represent any string. [ISO/IEC
167 14360 {2}]

168 **2.2.2.28 minimally consistent object:** An object that satisfies various condi-
169 tions set forth in the definition of its class. [ISO/IEC 14360 {2}]

170 **2.2.2.29 name:** An identifier used to represent a specific value of a specific
171 datatype.

172 This International Standard does not specify when a name is bound to its value: at
173 compile time, link time, or load time. The use of a name as a parameter in a data-
174 type specification does not imply that its value is defined at compile time.
175 [ISO/IEC 14360 {2}]

176 **2.2.2.30 object:** Any of the complex information objects created, examined,
177 modified, or destroyed by means of the interface. [ISO/IEC 14360 {2}]

178 **2.2.2.31 octet:** An ordered sequence of 8 b. [ISO/IEC 14360 {2}]

179 **2.2.2.32 octet string:** A string comprised of octets. [ISO/IEC 14360 {2}]

180 **2.2.2.33 operation:** A program abstraction with formal input and output param-
181 eters that are bound to objects or values in the application that invokes it.
182 [ISO/IEC 14360 {2}]

2.2.2.34 OSI abstract data manipulation application program interface:

The interface defined by this International Standard and ISO/IEC 14360 {2}.

2.2.2.35 package: A group of related classes. [ISO/IEC 14360 {2}]

2.2.2.36 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. [ISO/IEC 14360 {2}]

2.2.2.37 position (within a string): The ordinal position of one element of a string relative to another. [ISO/IEC 14360 {2}]

2.2.2.38 position (within an attribute): The ordinal position of one value relative to another. [ISO/IEC 14360 {2}]

2.2.2.39 primary representation: The form in which the service supplies an attribute value to the client. [ISO/IEC 14360 {2}]

2.2.2.40 private object: The internal representation of the language-independent model of an object in a service, and thus unspecified. [ISO/IEC 14360 {2}]

2.2.2.41 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. [ISO/IEC 14360 {2}]

2.2.2.42 public object: The representation by the client of an object in a particular language binding. [ISO/IEC 14360 {2}]

2.2.2.43 secondary representation: A second form, an alternative to the primary representation, in which the client may supply an attribute value to the service. [ISO/IEC 14360 {2}]

2.2.2.44 segment: Zero or more contiguous elements of a string. [ISO/IEC 14360 {2}]

2.2.2.45 service: Software that implements the interface. [ISO/IEC 14360 {2}]

- 211 **2.2.2.46 service interface:** The interface as realized, for the benefit of the client,
by the service as a whole. [ISO/IEC 14360 {2}]
- 212 **2.2.2.47 specific:** With respect to a class, the attribute types that may appear in
213 an instance of the class but not in an instance of its superclasses. [ISO/IEC 14360
214 {2}]
- 215 **2.2.2.48 status datatype:** An abstract datatype whose values may be bound to
216 “control” values as well as “data” values. [ISO/IEC 14360 {2}]
- 217 **2.2.2.49 string:** An ordered sequence of zero or more bits, octets, or characters,
218 accompanied by the length of the string. [ISO/IEC 14360 {2}]
- 219 **2.2.2.50 subclass:** One of the classes, designated as such, whose attribute types
220 are a superset of those of another class. [ISO/IEC 14360 {2}]
- 221 **2.2.2.51 subobject:** An immediate subobject of an object or of one of its subobjects
222 applied recursively. [ISO/IEC 14360 {2}]
- 223 **2.2.2.52 superclass:** One of the classes, designated as such, whose attribute
224 types are a subset of those of another class. [ISO/IEC 14360 {2}]
- 225 **2.2.2.53 superobject:** The immediate superobject of an object, or one of its
226 superobjects applied recursively. [ISO/IEC 14360 {2}]
- 227 **2.2.2.54 syntax:** A category into which an attribute value is placed on the basis of
228 its form. [ISO/IEC 14360 {2}]
- 229 **2.2.2.55 syntax template:** A lexical construct containing an asterisk from which
230 several attribute syntaxes can be derived by substituting text for the asterisk.
231 [ISO/IEC 14360 {2}]
- 232 **2.2.2.56 test methods:** The software, procedures, or other means specified to
233 measure conformance to a specification. [ISO/IEC 14360 {2}]
- 234 **2.2.2.57 type:** A category into which attribute values are placed on the basis of
235 their purpose. [ISO/IEC 14360 {2}]
- 236 **2.2.2.58 value:** An arbitrarily complex information item that can be viewed as a
237 characteristic or property of an object. [ISO/IEC 14360 {2}]

2.2.2.59 workspace: A repository for instances of classes in the closures of one or more packages associated with the workspace. [ISO/IEC 14360 {2}]

2.2.2.60 workspace interface: The interface as realized, for the benefit of the dispatcher, by each workspace individually.

2.2.3 Abbreviations

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

2.2.3.1 API: Application Program Interface.

2.2.3.2 ASN.1: Abstract Syntax Notation One (defined in ISO/IEC 8824 {B1}).

2.2.3.3 BER: Basic Encoding Rules (defined in ISO/IEC 8825 {B2}).

2.2.3.4 C Standard: ISO/IEC 9899: 1990, *Information technology—Programming languages—C* {1}.

2.2.3.5 IA5: International Alphabet No. 5 (used in “IA5 String”; see ISO/IEC 8824 {B1}).

2.2.3.6 OM: OSI abstract data Manipulation.

2.2.3.7 OSI: Open Systems Interconnection.

2.2.3.8 UTC: Universal Coordinated Time.

Section 3: Service Interface

3.1 Introduction

This section defines the C language binding to the service interface.

3.2 Datatypes

This section defines the datatypes of the C language binding to the service interface.

3.2.1 Intermediate Datatypes

Some datatypes are defined in terms of the following “intermediate datatypes,” whose precise definitions in C are system defined:

<i>OM_sint</i>	The positive and negative integers that can be represented in 16 b.
<i>OM_sint16</i>	The positive and negative integers that can be represented in 16 b.
<i>OM_sint32</i>	The positive and negative integers that can be represented in 32 b.
<i>OM_uint</i>	The nonnegative integers that can be represented in 16 b.
<i>OM_uint16</i>	The nonnegative integers that can be represented in 16 b.
<i>OM_uint32</i>	The nonnegative integers that can be represented in 32 b.
<i>OM_double</i>	The positive and negative floating point numbers representable in 64 b.

The range of integers for *OM_uint* shall encompass the range for *OM_sint*.

NOTE: The *OM_sint* and *OM_uint* datatypes are defined above by the ranges of integers they have to accommodate. Implementations can define them by the range of integers the word size of the host machine permits. The *OM_double* datatype is defined by the range and precision of real numbers that it has to accommodate. The following examples will work on most machines.

```

typedef int          OM_sint;
26 typedef int          OM_sint16;
27 typedef long int    OM_sint32;
28 typedef unsigned    OM_uint;
29 typedef unsigned    OM_uint16;
30 typedef long unsigned OM_uint32;
31 typedef double       OM_double;

```

32 3.2.2 Boolean

33 3.2.2.1 Name

34 Boolean—Type definition for a Boolean data value.

35 3.2.2.2 C Declaration

```
36 typedef OM_uint32 OM_boolean;
```

37 NOTE: The type is defined as 32 b in order to facilitate aggregate initialization of the *OM_value*
38 union.

39 3.2.2.3 Description

40 A data value of this datatype is a Boolean, i.e., either **false** or **true**.

41 In the C interface, false is denoted by zero {OM_FALSE}, and true is denoted by
42 any other integer (although the symbolic constant {OM_TRUE} refers to the
43 integer one specifically).

44 3.2.3 Descriptor

45 3.2.3.1 Name

46 Descriptor—Type definition for describing an attribute type and value.

47 3.2.3.2 C Declaration

```

48 typedef struct OM_descriptor_struct
49 {
50     OM_type      type;
51     OM_syntax     syntax;
52     OM_value      value;
53 } OM_descriptor;

```

54 NOTE: Other components are encoded in the syntax member.

3.2.3.3 Description

A data value of this datatype is a descriptor, which embodies an attribute value. A sequence of descriptors (an array in C) can represent all the values of all the attributes of an object, and is the representation called a **Public Object**. A descriptor has the following components:

type (Type) Identifies the type of the attribute value.

syntax (Syntax)

Identifies the syntax of the attribute value and contains additional information, as described below.

value (Value)

The attribute value.

Additional information is encoded in the *syntax* structure member. The syntax should always be masked with the constant {OM_S_SYNTAX} because of this. For example:

```
my_syntax = my_public_object[3].syntax & OM_S_SYNTAX;
my_public_object[4].syntax =
my_syntax + (my_public_object[4].syntax & ~OM_S_SYNTAX);
```

Alternatively, the macros OM_SYNTAX(d) and OM_SYNTAX_ASSIGN(d,s) may be used for this purpose:

```
my_syntax = OM_SYNTAX( my_public_object[3]) ;
OM_SYNTAX_ASSIGN( my_public_object[4], my_syntax) ;
```

The additional information can be obtained by masking the syntax with the following constants:

{OM_S_LONG_STRING}

True (i.e., nonzero) if, and only if, the descriptor is service generated and the length of the value is greater than an implementation-defined limit.

The macro OM_IS_LONG_STRING(d) may be used to test this flag.

{OM_S_NO_VALUE}

Only true if the descriptor is service generated and the value is not present because {OM_EXCLUDE_VALUES} or {OM_EXCLUDE_MULTIPLES} was set in the call to *om_get()*.

The macro OM_HAS_VALUE(d) may be used to test this flag.

{OM_S_LOCAL_STRING}

Only significant if the Syntax is String(*). True if, and only if, the string is represented in an implementation-defined local character set. The local character set may be more amenable for use as keyboard input or display output than the nonlocal character set, and it may include specific treatment of line termination sequences. Certain interface functions may convert information in string syntaxes to or from the local representation, which may result in a loss of information.