

IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation C Language Interfaces—Binding for Application Program Interface (API)

Sponsor

**Portable Applications Standards Committee
of the
IEEE Computer Society**

Approved March 18, 1993

IEEE Standards Board

Abstract: A C programming language binding for the general-purpose OSI Abstract Data Manipulation (OM) Application Program Interface (API) in IEEE Std 1224-1993 is defined. This standard is intended for use by application developers, system implementors, test method writers, and users to support application portability at the source-code level. The API defined in this standard contains two major parts: an interface for accessing the services defined in the programming language independent specification, and an interface for language-specific services.

Keywords: abstract syntax notation, API, application portability, application program interface, ASN.1, C language, language binding, 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 1-55937-311-3

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

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

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

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of 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.

Introduction

(This introduction is not a normative part of IEEE Std 1327-1993, IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation C Language Interfaces—Binding for Application Program Interface (API), but is included for information only.)

Purpose

The purpose of this standard is to define a C programming language binding for the general-purpose OSI Abstract Data Manipulation (OM) Application Programming Interface (API) that is defined in IEEE Std 1224-1993 {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.

Related Standards

The programming language independent specification on which the programming language binding defined in this standard is based is contained in IEEE Std 1224-1993 {2}.

This 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 IEEE Std 1327.1-1993 {B6} and the API to Directory Services defined in IEEE Std 1327.2-1993 {B7}.

IEEE Std 1328-1993 {4} specifies a set of requirements that need to be satisfied for measuring conformance to this standard. In addition, test methods must also satisfy the requirements specified in IEEE Std 1326-1993 {3} for measuring conformance to the programming language-independent specification of the API.

Overview

The API defined in this 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 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 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

Extensions are approved as “amendments” or “revisions” to this document, following IEEE Standards procedures.

Approved amendments are published separately and distributed with orders from the IEEE for this document until the full document is reprinted and such amendments are incorporated in their proper positions.

¹A *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.

If you have any question about the completeness of your version, you may contact the IEEE Standards Department [+1 (908) 562-3800] to determine what supplements have been published.

If you have an interest in participating in the working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA, and ask to have this forwarded to the chairperson of the appropriate PASC working group

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)

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

P1224 Working Group Officials

Steve Trus, *Chair*
Tim Carter, *Vice Chair*
Graham Jack, *Editor/Secretary*

Working Group

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

Petr Janecek
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 the standard for submission to the IEEE Standards Board:

Ralph Barker, *Uniform Organizational Representative*
Petr Janecek, *X/Open Organizational Representative*

David Athersych
Ralph Barker
Peter Beaman
Andy Bihain
Mack W. Bishop
Tim Carter
Andy Cheese
Kilnam Chon
Ian Devine
Philip H. Enslow

K. Frimpong-Ansah
Michel Gien
Steve Griesmer
Scott Guthery
Charles Hammons
C. J. Harding
Mark Hazzard
James Hegerty
Niraj Jain
Hal Jespersen

Derek Kaufman
Norman Kincl
Joseph King
Stephen Layne
Joachim Loitzenbauer
Steven Lucas
Joseph F. P. Luhukay
Roger Martin
Joberto S. B. Martins
John S. Morris

Andrew L. Nicholson
Jan Osterweil
Edward Owens
Ramesh Pabbati
Prayoon Phathayakorn
Dave Plauger
Udo W. Pooch
Scott E. Preece
Paul Rabin
Steve Rago

Joseph Ramus
Robert Robillard
Wolfgang Schwabl
Richard Seibel
Dan Shia
Jim Sides
John P. Slone
James G. Tanner
Alec Tashjian
Ravi Tavakley

Donn S. Terry
Trevor J. Thompson
Mark-Rene Uchida
Paul Wanish
Bruce Weiner
Les Wiberly
Peter Wishart
Oren Yuen
George R. Zerdian
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
Clyde R. Camp
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ramiro Garcia
Donald N. Heirman
Jim Isaak

Ben C. Johnson
Walter J. Karplus
Lorraine C. Kevra
E. G. "Al" Kiener
Ivor N. Knight
Joseph Koepfinger*
D. N. "Jim" Logothetis
Don T. Michael*

Marco W. Migliaro
L. John Rankine
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp
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

CLAUSE	PAGE
1. General	1
1.1 Scope	1
1.2 Normative References	1
1.3 Conformance	2
1.4 Test Methods	3
2. Terminology and General Requirements	3
2.1 Conventions	3
2.2 Definitions	4
3. Service Interface	8
3.1 Introduction	8
3.2 Datatypes	8
3.3 Functions	22
3.4 Return Codes	26
3.5 Header File	27
4. Workspace Interface	27
4.1 Introduction	27
4.2 Representation of Objects	27
4.3 Types and Macros	28
4.4 Dispatcher Macros	32
5. Definitions of Constants	33
5.1 Exclusions	33
5.2 Modification	33
5.3 Object Identifiers	34
5.4 Return Code	34
5.5 String (Elements Component)	34
5.6 Syntax	34
5.7 Type	34
Annex A (Informative) Example $\langle xom.h \rangle$ Header	37
Annex B (Informative) Bibliography	44

IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation C Language Interfaces—Binding for Application Program Interface (API)

1. General

1.1 Scope

This standard defines an ISO 9989 :1990 {1}¹C language binding to the language-independent API defined in IEEE Std 1224–1993 {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 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 9899:1990,² *Programming Languages—C*.

{2}IEEE Std 1224–1993,³ *IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation—Application Program Interface (API) [Language Independent]*.

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

² ISO/IEC documents can be obtained from the ISO office, 1 Rue de Varembe, Case Postale 56, 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.

{3}IEEE Std 1326–1993, *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to Open Systems Interconnection (OSI) Abstract Data Manipulation—Application Program Interface (API) [Language Independent]*.

{4}IEEE Std 1328–1993, *IEEE Standard for 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 standard shall be a conforming implementation, as defined in IEEE Std 1224-1993 {2}, for the programming language binding specification that is contained in this standard.

1.3.1.2 Documentation

A conformance document shall be available for an implementation claiming conformance to this standard. It shall satisfy the requirements that are defined in IEEE Std 1224-1993 {2} for a conformance document for a programming language binding specification.

The phrases “shall document” or “shall be documented” in this 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 IEEE Std 1224-1993 {2}, the following aspects of the behavior of the service are implementation defined:

- 1) Whether it supports interworking with other implementations
- 2) The precise definitions in C of the intermediate datatypes

The conformance document shall state whether the implementation supports interworking with other implementations. If it supports interworking with other implementations, then the implementation shall satisfy the requirements of Sections 4 (Workspace Interface) and 5 (Definitions of Symbolic Constants) of this standard.

1.3.2 Application Conformance

All applications claiming conformance to this 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 standard is a Strictly Conforming Application as defined in IEEE Std 1224-1993 {2} for the programming language binding specification contained in this 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 standard is an ISO/IEC Conforming Application as defined in IEEE Std 1224-1993 {2} for the programming language binding specification contained in this standard.

1.3.2.3 Conforming Application Using Extensions

A Conforming Application Using Extensions for this standard is a Conforming Application Using Extensions as defined in IEEE Std 1224-1993 {2} for the programming language binding specification contained in this standard.

1.4 Test Methods

Any measurement of conformance to this standard shall be performed using test methods that conform to IEEE Std 1326-1993 {3} and to IEEE Std 1328-1993 {4}.

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 the 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 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 :1990 {1}), and the *public namespace* is the set of all possible public identifiers.

Table 2.1—Naming Conventions

Item	Prefix
reserved for implementors reserved for interface extensions reserved for implementors reserved for interface extensions	omP omX OMP OMX
functions OM class names other constants	om_ OM_C_ 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 IEEE Std 1224-1993 {2} for further details on the use of object identifiers.

2.1.3 Language-Independent Conventions

The language-independent conventions defined in IEEE Std 1224 -1993 {2} shall apply in this standard.

2.2 Definitions

2.2.1 Terminology

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

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

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

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 standard, but that can be provided. With respect to Strictly Conforming Applications, the word *may* means that the optional feature shall not be used. [ISO/IEC 9945-1 :1990 {B1}]

2.2.1.4 shall: An indication of a requirement on the implementation or on Strictly Conforming Applications, where appropriate. [ISO/IEC 9945-1 :1990 {B1}]

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 :1990 {B1}]

2.2.1.6 supported: A condition regarding optional functionality.

Certain functionality in this standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified by this 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 :1990 {B1}]

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 :1990 {B1}]

2.2.1.8 undefined: An indication that this 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 :1990 {B1}]

2.2.1.9 unspecified: An indication that this 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 :1990 {B1}]

2.2.2 General Terms

For the purposes of this 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. [IEEE Std 1224-1993 {2}]

2.2.2.2 accessible: An object for which the client possesses a valid designator or handle [IEEE Std 1224-1993 {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. [IEEE Std 1224-1993 {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. [IEEE Std 1224-1993 {2}]

2.2.2.5 bit string: An ordered sequence of zero or more bits. [IEEE Std 1224-1993 {2}]

2.2.2.6 byte: A unit of machine storage containing an ordered sequence of 8 b. [IEEE Std 1224-1993 {2}]

2.2.2.7 C interface: The C language binding, defined in terms of ISO/IEC 9899 : 1990 {1}

2.2.2.8 character string: An ordered sequence of zero or more characters. [IEEE Std 1224-1993 {2}]

2.2.2.9 class: A category into which objects are placed on the basis of both their purpose and their internal structure. [IEEE Std 1224-1993 {2}]

2.2.2.10 client: Software that uses the interface. [IEEE Std 1224-1993 {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. [IEEE Std 1224-1993 {2}]

2.2.2.12 datatype: A collection of distinguished values, together with a collection of characterizing operations on those values. [IEEE Std 1224-1993 {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. [IEEE Std 1224-1993 {2}]

2.2.2.16 generic interface: The interface, defined at a level that is independent of any particular programming language. [IEEE Std 1224-1993 {2}]

2.2.2.17 immediate subclass: A subclass, of a class C, having no superclasses that are themselves subclasses of C. [IEEE Std 1224-1993 {2}]

2.2.2.18 immediate subobject: One object that is a value of an attribute of another. [IEEE Std 1224-1993 {2}]

2.2.2.19 immediate superclass: The superclass, of a class C, having no subclasses that are themselves superclasses of C. [IEEE Std 1224-1993 {2}]

2.2.2.20 immediate superobject: One object that contains another among its attribute values. [IEEE Std 1224-1993 {2}]

2.2.2.21 inaccessible: An object for which the client does not possess a valid designator or handle. [IEEE Std 1224-1993 {2}]

2.2.2.22 independent copy: A copy of the object plus independent copies of all its subobjects (applied recursively). [IEEE Std 1224-1993 {2}]

2.2.2.23 instance: An object in the category represented by a class. [IEEE Std 1224-1993 {2}]

2.2.2.24 interface: An abbreviation for the definition in 2.2.2.34.

2.2.2.25 intermediate datatype: Any of the basic datatypes in terms of which the other, substantive datatypes of the interface are defined. [IEEE Std 1224-1993 {2}]

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

2.2.2.27 Length:: The number of bytes required to represent any string. [IEEE Std 1224-1993 {2}]

2.2.2.28 minimally consistent object: An object that satisfies various conditions set forth in the definition of its class. [IEEE Std 1224-1993 {2}]

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

This 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. [IEEE Std 1224-1993 {2}]

2.2.2.30 object: Any of the complex information objects created, examined, modified, or destroyed by means of the interface. [IEEE Std 1224-1993 {2}]

2.2.2.31 octet: An ordered sequence of 8 b. [IEEE Std 1224-1993 {2}]

2.2.2.32 octet string: A string comprised of octets. [IEEE Std 1224-1993 {2}]

2.2.2.33 operation: A program abstraction with formal input and output parameters that are bound to objects or values in the application that invokes it. [IEEE Std 1224-1993 {2}]

2.2.2.34 OSI abstract data manipulation application program interface: The interface defined by this standard and IEEE Std 1224-1993 {2}.

2.2.2.35 package: A group of related classes. [IEEE Std 1224-1993 {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. [IEEE Std 1224-1993 {2}]

2.2.2.37 position (within a string): The ordinal position of one element of a string relative to another. [IEEE Std 1224-1993 {2}]

2.2.2.38 position (within an attribute): The ordinal position of one value relative to another. [IEEE Std 1224-1993 {2}]

2.2.2.39 primary representation: The form in which the service supplies an attribute value to the client. [IEEE Std 1224-1993 {2}]

2.2.2.40 private object: The internal representation of the language-independent model of an object in a service, and thus unspecified. [IEEE Std 1224-1993 {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. [IEEE Std 1224-1993 {2}]

2.2.2.42 public object: The representation by the client of an object in a particular language binding. [IEEE Std 1224-1993 {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. [IEEE Std 1224-1993 {2}]

2.2.2.44 segment: Zero or more contiguous elements of a string. [IEEE Std 1224-1993 {2}]

2.2.2.45 service: Software that implements the interface. [IEEE Std 1224-1993 {2}]

2.2.2.46 service interface: The interface as realized, for the benefit of the client, by the service as a whole. [IEEE Std 1224-1993 {2}]

2.2.2.47 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. [IEEE Std 1224-1993 {2}]

2.2.2.48 status datatype: An abstract datatype whose values may be bound to “control” values as well as “data” values. [IEEE Std 1224-1993 {2}]

2.2.2.49 string: An ordered sequence of zero or more bits, octets, or characters, accompanied by the length of the string. [IEEE Std 1224-1993 {2}]

2.2.2.50 subclass: One of the classes, designated as such, whose attribute types are a superset of those of another class. [IEEE Std 1224-1993 {2}]

2.2.2.51 subobject: An immediate subobject of an object or of one of its subobjects applied recursively. [IEEE Std 1224-1993 {2}]

2.2.2.52 superclass: One of the classes, designated as such, whose attribute types are a subset of those of another class. [IEEE Std 1224-1993 {2}]

2.2.2.53 superobject: The immediate superobject of an object, or one of its superobjects applied recursively. [IEEE Std 1224-1993 {2}]

2.2.2.54 syntax: A category into which an attribute value is placed on the basis of its form. [IEEE Std 1224-1993 {2}]

2.2.2.55 syntax template: A lexical construct containing an asterisk from which several attribute syntaxes can be derived by substituting text for the asterisk. [IEEE Std 1224-1993 {2}]

2.2.2.56 test methods: The software, procedures, or other means specified to measure conformance to a specification. [IEEE Std 1224-1993 {2}]

2.2.2.57 type: A category into which attribute values are placed on the basis of their purpose. [IEEE Std 1224-1993 {2}]

2.2.2.58 value: An arbitrarily complex information item that can be viewed as a characteristic or property of an object. [IEEE Std 1224-1993 {2}]

2.2.2.59 workspace: A repository for instances of classes in the closures of one or more packages associated with the workspace. [IEEE Std 1224-1993 {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 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 :1990 {BS}).

2.2.3.3 BER: Basic Encoding Rules (defined in ISO/IEC 8825 :1990 {B9}).

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 :1990 {BS}).

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.

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:

OM_sint The positive and negative integers that can be represented in 16 b.
OM_sint16 The positive and negative integers that can be represented in 16 b.
OM_sint32 The positive and negative integers that can be represented in 32 b.
OM_uint The nonnegative integers that can be represented in 16 b.
OM_uint16 The nonnegative integers that can be represented in 16 b.
OM_uint32 The nonnegative integers that can be represented in 32 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 following examples will work on most machines.

```

typedef int          OM_sint;
typedef int          OM_sint16;
typedef long int     OM_sint32;
typedef unsigned     OM_uint;
typedef unsigned     OM_uint16;
typedef long unsigned OM_uint32;
  
```

3.2.2 Boolean

3.2.2.1 Name

Boolean—Type definition for a Boolean data value.

3.2.2.2 C Declaration

```
typedef OM_uint32 OM_boolean;
```

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

3.2.2.3 Description

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

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

3.2.3 Descriptor

3.2.3.1 Name

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

3.2.3.2 C Declaration

```

typedef struct OM_descriptor_struct
{
    OM_type      type;
  
```