

IEEE Standard for Information Technology—X.400-Based Electronic Messaging—Application Program Interface (API) [Language Independent]

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

Approved March 18, 1993
IEEE Standards Board

Approved September 16, 1993
American National Standards Institute

Abstract: Application program interfaces (APIs) to X.400-based electronic messaging services are defined in terms that are independent of any particular programming language. A general-purpose API that makes the functionality of a message transfer system (MTS) accessible to a message store (MS) or user agent (UA), or the functionality of a simple MS accessible to a UA, is provided. An X.400 gateway API divides a message transfer agent (MTA) into two software components, a messaging system gateway and an X.400 gateway service. This standard, and the language bindings derived from it, are intended to be used in conjunction with IEEE Std 1224.1-1993, which provides a general-purpose API for the creation, examination, modification, and deletion of OSI information objects.

Keywords: API, application portability, electronic messaging, language independent, object management, open systems, open systems interconnection, OSI, X.400

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 1-55937-302-4

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 may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers, Inc. does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Introduction

(This introduction is not a normative part of IEEE Std 1224.1-1993, IEEE Standard for Information Technology—X.400-Based Electronic Messaging—Application Program Interface (API) [Language Independent], but is included for information only.)

Purpose

The purpose of this standard is to define an Application Program Interface (API) to X.400-based electronic messaging services in terms that are independent of any particular programming language. The APIs specified in this standard provide for access to, and interconnection of, messaging systems whose architecture is in accordance with X.400.

Related Standards

This standard is intended to provide the basis for the definition of programming language bindings to which implementations and applications can conform. Such a language binding, for the C programming language, is contained in IEEE Std 1327.1-1993 {B.4}.

This standard, and the language bindings derived from it, are intended to be used in conjunction with the OSI abstract data manipulation API, IEEE Std 1224-1993 {42}, which provides a general-purpose API for the creation, examination, modification, and deletion of OSI information objects.

IEEE Std 1326.1-1993 {44} contains a set of requirements to be satisfied by test methods for measuring conformance to this standard. They are stated in terms that are independent of any particular programming language, and they apply to test methods for measuring conformance to all standards defining programming language bindings to this standard.

Overview

This document defines two APIs of architectural importance in the construction of message handling systems (MHSs). It defines an X.400 API that makes the functionality of a message transfer system (MTS) accessible to a message store (MS) or user agent (UA), or the functionality of a simple MS accessible to a UA. It also defines an X.400 Gateway API that divides a message transfer agent (MTA) into two software components, a mail-system gateway and an X.400 gateway service. The interfaces are designed to be used and implemented in conjunction with the general-purpose OSI Abstract Data Manipulation API defined in IEEE Std 1224-1993 {42}.

The two interfaces support Message Access (MA), Message Transfer (MT), and Interpersonal Messaging (IM). MA is the conveyance of information objects between the MTS and one of its users, and thereby among its various users. MT is the conveyance of such objects between the MTAs that make up the MTS. IM is the MA and MT application in which the contents of messages are electronic memos. The interfaces address the operational requirements of these activities. They do not address their system management requirements; for example, those for security.

Acknowledgments

The base document for this standard is XO/CAE/91/100: 1991, the *API to Electronic Mail (X. 400)* {8}, copyright © 1989–1990 by X.400 API Association, United States of America, and by X/Open Company Limited, United Kingdom. The IEEE is grateful to both X/Open and the X.400 API Association for permission to use their materials.

Related Standards Activities

The following areas are under active consideration at this time, or are expected to become active in the near future, concerning extensions to this standard. Similar efforts can be anticipated in the future:¹

- 1) Directory services
- 2) FTAM API
- 3) Verification testing methods
- 4) Network interface facilities
- 5) System administration

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

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

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 amendments 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)

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

P1224 Working Group Officials

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

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: +1202371-0101; FAX: +1202728-9614.

P1224 Working Group

Curt Campbell
Tony Cincotta
Chris Harding
Khiem Ho
Petr JaneDCLeK

Mark Karan
Eva Kuiper
Kevin Montgomery
Ed Owens
Pierre Perrot

Enzo Signore
Daniella Sirocchi
Andrew Twigger
Bruce Weiner

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

	Stephen R. Walli (<i>EurOpen Organizational Representative</i>)	
	Ralph Barker (<i>Uniform Organizational Representative</i>)	
Timothy Baker	Steve Griesmer	Ronald K. Sellars
Ralph Barker	Charles Hammons	Sandra Swearingen
Andy Bihain	Michael Hannah	James G. Tanner
Mack W. Bishop	Mark Hazzard	Donn S. Terry
Robert Bismuth	Barry Hedquist	Alan E. Turner
George Carson	Khiem Ho	Mark-Rene Uchida
Tim Carter	Petr Janecek	Andrew Walker
Andy Cheese	Hal Jespersen	Paul Wanish
Fred D. Crouner	Robert Knighten	Peter Wishart
Ana Maria De Alvare	Kevin Lewis	Oren Yuen
Iain Devine	Roger Martin	George R. Zerdian
Stephen L. Diamond	Joberto S.B. Martins	Jason Zions
Philip H. Enslow	Diana Norwood	John Zolnowsky
Donald Folland	Edward Owens	
Michel Gien	Scott E. Preece	

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	4
1.4 Test Methods	7
2. Terminology and General Requirements	7
2.1 Conventions	7
2.2 Definitions	10
3. Overview	15
3.1 Purpose	15
3.2 Overview	16
3.3 Object Management	16
3.4 Features	20
4. Message Handling Interfaces	21
4.1 Datatypes	21
4.2 Access Operations	22
4.3 Transfer Operations	33
4.4 Return Codes	41
5. Interpersonal Messaging Packages	45
5.1 Class Hierarchy	45
5.2 Class Definitions	46
5.3 Syntax Definitions	58
6. Message Handling Packages	60
6.1 Class Hierarchy	60
6.2 Class Definitions	61
6.3 Syntax Definitions	109
7. Secure Messaging Package	114
7.1 Class Hierarchy	114
7.2 Class Definitions	115
7.3 Syntax Definitions	120
8. Definition of Constants	120
8.1 Object Identifier for This Standard	120
8.2 OIDs for Features and Packages	120
8.3 Interpersonal Messaging Package Value Definitions	121
8.4 Message Handling Package Value Definitions	123
8.5 Secure Messaging Package Value Definitions	126

CLAUSE	PAGE
Annex A Functional Architecture of an API (Informative)	128
Annex B An Introduction to X.400 (Informative)	134
Annex C Gateway Considerations (Informative).....	137
Annex D Bibliography (Informative).....	139

IEEE Standard for Information Technology—X.400-Based Electronic Messaging—Application Program Interface (API) [Language Independent]

1. General

1.1 Scope

The purpose of this standard is to provide APIs supporting the development of applications that are users of the message transfer system, and gateways that incorporate or use X.400 mail functionality. This includes gateways between X.400 mail networks and proprietary mail systems. The interface supports access to X.400 services.

The standard defines two application program interfaces to X.400-based electronic messaging systems. The interfaces are

- 1) An X.400 API that makes the functionality of a message transfer system (MTS) accessible to a message store (MS) or user agent (UA), or the functionality of a simple MS accessible to a UA
- 2) An X.400 Gateway API that divides a message transfer agent (MTA) into two software components, a messaging system gateway and an X.400 gateway service

This standard provides a language-independent specification.

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 639: 1988, *Codes for the representation of names of languages*.¹
- [2] ISO 3166: 1988, *Codes for the representation of names of countries*.
- [3] ISO 6937-1: 1983, *Information processing—Coded character sets for text communication—Part 1: General introduction*.
- [4] ISO 6937-2: 1983, *Information processing—Coded character sets for text communication—Part 2: Latin alphabetic and non-alphabetic graphic characters*.
- [5] ISO/IEC 7498:1984 (CCITT Recommendation X. 200: 1984), *Information processing systems—Open Systems Interconnection—Basic Reference Model*.
- [6] ISO 8883 ..., *Information Processing (Text Communication) Message-Oriented Text Interchange System, Message Transfer Sublayer, Message Interchange Service and Message Transfer Protocol*.
- [7] ISO 9065..., *Information Processing - Text Communication - Message Oriented Text Interchange System User Agent Sublayer - Inter-personal messaging user agent - Message interchange formats and protocols*.
- [8] ISO/IEC 10021-1:1990 (CCITT Recommendation X.400:1988, CCITT Recommendation X.408:1988), *Information technology—Text Communication—Message-Oriented Text Interchange Systems (MOTIS)—Part 1: System and Service Overview*.
- [9] ISO/IEC 10021-2:1990 (CCITT Recommendation X.402:1988), *Information technology—Text Communication—Message-Oriented Text Interchange Systems (MOTIS)—Part 2: Overall Architecture*.
- [10] ISO/IEC 10021-3:1990 (CCITT Recommendation X.407:1988), *Information technology—Text Communication—Message-Oriented Text Interchange Systems (MOTIS)—Part 3: Abstract Service Definition Conventions*.
- [11] ISO/IEC 10021-4:1990 (CCITT Recommendation X.411:1988), *Information technology—Text Communication—Message-Oriented Text Interchange Systems (MOTIS)—Part 4: Message Transfer System: Abstract Service Definition and Procedures*.
- [12] ISO/IEC 10021-5:1990 (CCITT Recommendation X.413:1988), *Information technology—Text Communication—Message-Oriented Text Interchange Systems (MOTIS)—Part 5: Message Store: Abstract Service Definition*.
- [13] ISO/IEC 10021-6:1990 (CCITT Recommendation X.419:1988), *Information technology—Text Communication—Message-Oriented Text Interchange Systems (MOTIS)—Part 6: Protocol Specifications*.
- [14] ISO/IEC 10021-7:1990 (CCITT Recommendation X.420:1988), *Information technology—Text Communication—Message-Oriented Text Interchange Systems (MOTIS)—Part 7: Interpersonal Messaging System*.
- [15] CCITT Recommendation E.164:1991, *Numbering plan for the ISDN era*.²
- [16] CCITT Recommendation F.1: 1984, *Operational provisions for the international public telegram service*.
- [17] CCITT Recommendation F.170: 1988, *Operational provisions for the international public facsimile service between public bureau (bureaufax)*.

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

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

- [18] CCITT Recommendation F.200: 1984, *Teletex service*.
- [19] CCITT Recommendation T.4: 1984, *Standardisation of group 3 facsimile apparatus for document transmission*.
- [20] CCITT Recommendation T.5: 1984, *General aspects of group 4 facsimile apparatus*.
- [21] CCITT Recommendation T.6: 1984, *Facsimile coding schemes and coding control functions for group 4 facsimile apparatus*.
- [22] CCITT Recommendation T.30: 1984, *Procedures for document facsimile transmission in the general switched telephone network*.
- [23] CCITT Recommendation T.50: 1984, *International Reference Alphabet*.
- [24] CCITT Recommendation T.60: 1984, *Terminal equipment for use in the teletex service*.
- [25] CCITT Recommendation T.61: 1984, *Character repertoire and coded character sets for the international teletex service*.
- [26] CCITT Recommendation T.62: 1984, *Control procedures for teletex and group 4 facsimile services*.
- [27] CCITT Recommendation T. 100: 1984, *International information exchange for interactive videotex*.
- [28] CCITT Recommendation T.101: 1984, *International interworking for videotex services*.
- [29] CCITT Recommendation T.400: 1984, *Introduction to document architecture, transfer, and manipulation*.
- [30] CCITT Recommendation T.411: 1988, *Open document architecture (ODA) and interchange format—introduction and general principles*.
- [31] CCITT Recommendation T.501: 1984, *A document application profile MM for the interchange of formatted mixed-mode documents*.
- [32] CCITT Recommendation T.503: 1984, *A document application profile for the interchange of Group 4 facsimile documents*.
- [33] CCITT Recommendation X.121: 1984, *International numbering plan for public data networks*.
- [34] CCITT Recommendation X.400:1984, *Message handling systems: System model, service elements*.
- [35] CCITT Recommendation X.401:1984, *Message handling systems: Basic service elements and optional user facilities*.
- [36] CCITT Recommendation X.408:1984, *Message handling systems: Encoded information type conversion rules*.
- [37] CCITT Recommendation X.409:1984, *Message handling systems: Presentation transfer syntax and notation*.
- [38] CCITT Recommendation X.410:1984, *Message handling systems: Remote operations and reliable transfer service*.
- [39] CCITT Recommendation X.411:1984, *Message handling systems: Message transfer layer*.
- [40] CCITT Recommendation X.420:1984, *Message handling systems: Interpersonal messaging user agent layer*.

[41] CCITT Recommendation X.430:1984, *Message handling systems: Access protocol for teletex terminals*.

[42] CEN/CENELEC ENV 41 202, *Message Handling Systems; User Agent (UA) and Message Transfer Agent (MTA): Access to an Administration Management Domain (ADMD)*, August 1987.³

[43] IEEE Std 1224-1993, *IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation—Application Program Interface (API) [Language Independent]*.⁴

[44] IEEE Std 1224.2-1993, *IEEE Standard for Information Technology—Directory Services—Application Program Interface (API) [Language Independent]*.

[45] IEEE Std 1326.1-1993, *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to X.400-Based Electronic Messaging—Application Program Interface (API) [Language Independent]*.

[46] *Stable Implementation Agreements for Open System Interconnection Protocols*, Version 4, Edition 1. Gaithersburg, MD: National Institute of Standards and Technology (NIST), 1990.⁵

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Conformance Requirements

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

- 1) **Support of defined behavior.** The implementation shall support all required behavior defined in this standard.
- 2) **Support of nonstandard extensions.** The implementation may provide additional operations or facilities not required by this standard or by the programming language binding specification. Each such nonstandard extension shall be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of operations or facilities defined by this 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 standard and the programming language binding specification. In no case shall such an environment require modification of a Strictly Conforming Application.
- 3) **API requirements.** The manufacturer and product shall satisfy the following requirements related to APIs:
 - The manufacturer shall identify the APIs (MA, MT, or both) the product implements and state what roles (client, service, or both) it plays for each. Those APIs shall support the behavior described in this standard and in the programming language binding specification.
 - The product shall implement the OM API defined in IEEE Std 1224-1993 [43],⁶ satisfying its conformance requirements, and implement the same roles for that API as it implements for the MA and MT interfaces.
- 4) **Features.** The manufacturer and product shall satisfy the following requirements related to features:
 - If the product implements the role of service for the MA or MT interface, the manufacturer shall state which features it implements.
 - If the product implements the role of service for the MA interface, it shall implement the Basic Access Functional Unit (FU) and the Submission FU, the Delivery FU, the Retrieval FU, or any combination of these.

³CEN/CENLEC publications can be obtained from the CEN/CENLEC, 2 Rue Brederode Bte.5, B-1000 Brussels, Belgium.

⁴IEEE documents can be obtained from the Institute of Electrical and Electronic Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ, 08855-1331, USA.

⁵NIST publications can be obtained from the Superintendent of Documents, US Government Printing Office, P.O. Box 37082, Washington, DC 20013-7082, USA.

⁶The numbers in curly brackets correspond to those of the references in 1.2.

- If the product plays the role of service for the MT interface, it shall implement the Basic Transfer FU and the Transfer In FU, the Transfer Out FU, or both.
 - If the product plays the role of service for either interface, the product shall implement the MH 88 Package, the MH 84 Package, or both.
- 5) **Operations.** The manufacturer and product shall satisfy the following requirement related to operations:
- The product shall implement every aspect of every operation in each FU for which it plays the role of service.
- 6) **Classes.** The manufacturer and product shall satisfy the following requirements related to classes:
- If the product implements the IM 88 or MH 88 Package, the manufacturer shall state which (if any) of its 1988 classes it implements.
 - If the product implements the IM 88 or IM 84 Package, the manufacturer shall state which (if any) immediate subclasses of the Body Part class the product implements.
 - If the product implements the IM 84 or SM 88 Package, it shall implement every class the package contains.
 - If the product implements the MH 84 Package, it shall implement all of those classes that are used by the functional units that it supports.
 - The product shall implement the closures of all classes it implements.
 - The product shall state for which classes it provides the **Encode** and **Decode** operations. Message Submission and Delivery Protocol (P3) or Message Retrieval Protocol (P7) of X.400.
 - If the product plays the role of service for the MT interface, the manufacturer shall state whether or not it realizes the interface by means of the Message Transfer Protocol (P1) of X.400.
 - If the product plays the role of service for either interface and implements the IM 88 or IM 84 Package, the manufacturer shall state whether or not the product realizes the interface by means of the Interpersonal Messaging Protocol (P2) of X.400.
 - If the product implements P1 (1988), P3 (1988), or P7 (1988), the manufacturer and product shall satisfy the conformance requirements of X.400 (1988) with respect to those protocols.
- NOTE — A 1988 package can be realized by means of the corresponding 1988 protocol alone, but a 1984 package can be realized using either the 1984 or the 1988 protocol.
- 8) **Options.** The manufacturer and product shall satisfy the following requirement related to implementation options:
- If the product plays the role of service for the MA or MT interface, the manufacturer shall state the behavior of implementation-defined options.
 - *Implementation of optional constant definitions.* Section 8. contains a set of definitions of values for the OM class object identifiers and OM attribute type identifiers used in this standard. Use of these definitions by an implementation is optional. *However*, if an implementation is to interwork with other implementations of this standard or of related standards (such as the OSI Abstract Data Manipulation API defined in IEEE Std 1224-1993 [43]) then it *shall* use the definitions given in Section 8. of this standard.
- 9) **Interpretation of any syntax.** Wherever an *any* appears in the syntax column of an attribute definition, this shall be treated as the corresponding OM syntax wherever the underlying ASN.1 encoding is a Universal simple type, as listed in IEEE Std 1224-1993 [43].

1.3.1.2 Conformance Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to a programming language binding specification for this standard. The conformance document shall be in two parts. The first part shall have the same structure as this 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 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 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 standard and in the programming language binding specification. This requirement shall be met by listing these features and by providing either a specific reference to the system documentation or full syntax and semantics of these features. The conformance document may specify the behavior of the implementation for those features where this standard or the programming language binding specification states that implementations may vary or where features are identified as undefined or unspecified.

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

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

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

1.3.1.3 Conforming Implementation Options

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

- 1) The circumstances that define abnormal termination of a session
- 2) The features of a session that the `mx_open` operation supplies as defaults
- 3) Whether the service will accept any circumstances as causes of temporary delivery failure
- 4) The maximum number of users that may be assigned to a single delivery queue
- 5) The maximum number of sessions that may exist simultaneously
- 6) Which qualifying object in the delivery or input queue (if there are several such objects) the `ma_start_delivery` or `mt_start_transfer_in` operation selects for delivery or transfer in
- 7) Which and how many sessions the `rex_wait` operation notifies of the arrival of an object when several sessions provide access to a delivery, retrieval, or input queue

1.3.2 Application Conformance

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

1.3.2.1 Strictly Conforming Application

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

1.3.2.2 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 standard is an application that uses only the facilities described in this 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 standard is an application that differs from a Conforming Application only in that it uses nonstandard facilities that are consistent with this 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 standard shall be performed using test methods that conform to IEEE Std 1326.1-1993 [45] and to any additional requirements that may be imposed by the programming language binding specification.

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 (e.g., **conversion-loss-prohibited**). The first letters of language-independent OM class and OM attribute names are capitalized (e.g., **Redirection-Record**).

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

The use of fonts in this standard is as follows.

- The Helvetica font is used for:
 - Language-independent operation names, such as `ma_start_delivery`
 - Language-independent datatype names, such as `mh_interval_type`
 - Language-independent error names, such as `feature_unavailable`
- The *italic* font is used for:
 - Language-independent operation arguments, such as *user_address*
 - The introduction of important terms
 - Cross-references in 2.2
- The **bold** font is used for:
 - Language-independent concrete OM class names, such as **Delivered-Report**
 - Language-independent OM attribute names, such as **Algorithm-Datum**
 - Language-independent OM attribute values, such as **conversion-loss-prohibited**
- The ***bold italic*** font is used for:
 - Language-independent abstract OM class names, such as ***Delivery-Report***
- The constant width (Courier) font is used for:
 - References to terms defined in standards upon which this standard is based

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 POSIX language-independent specifications conventions which are being developed in IEEE PASC [B6].

This subclause defines the notation used within this 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 standard may take other forms.

The key elements of the notation are *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 the **Descriptions** subclause, 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 $2^{16} - 1$ 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

```

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

```

2.1.2.3 Numbers

```

<number> ::= <digit> | <digit><number>
<signed-number> ::= [ - ] <number>

```

2.1.2.4 Datatypes

The following notation is used to specify a datatype:

Datatype: <datatype-name>
Definition: <datatype>
Description: ...

where

```

<datatype> ::= <datatype-name> | <datatype-definition>
<datatype-name> ::= <identifier>
<datatype-definition> ::= <atomic-type> | <constructed-type>
<atomic-type> ::= boolean_type [ <opaque-type> |
<state-type> | <integer-type>
<opaque-type> ::= opaque [ <size> ]
<state-type> ::= state [ <state-list> ]
<state-list> ::= <state-value> [ , <state-list> ]
<state-value> ::= <identifier>
<size> ::= <number>
<integer-type> ::= integer_type [ <minimum> , <maximum> ]
<minimum> ::= <signed-number>
<maximum> ::= <signed-number>
<constructed-type> ::= <array-type> | <record-type> |
<set-type> | <choice-type> | <list-type>
<array-type> ::= array <index-type> of <base-type>
<index-type> ::= <integer-type> | <state-type>
<base-type> ::= <datatype>
<record-type> ::= record [ <member-list> ]
<choice-type> ::= choice [ <member-list> ]
<set-type> ::= set [ <datatype> ]
<list-type> ::= list [ <datatype> ] [ max members <number> ]
<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

$\langle \text{parameter-name} \rangle ::= \langle \text{identifier} \rangle$

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 [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 :1990 [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 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 [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 :1990 [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 9945-1:1990 [B2]]

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 [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 9945-1:1990 [B2]]

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 [B2]]

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 [B2]]

2.2.2 General Terms

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

2.2.2.1 1988 class: A class for which all the specific attributes are for the 1988 version of the CCITT X.400 Recommendation alone.

2.2.2.2 abstract class: A class, instances of which are forbidden unless they belong to one of its concrete subclasses. [IEEE Std 1224-1993 [43]]

2.2.2.3 array: A datatype constructed from a base datatype and an index datatype. The base datatype may be any datatype; the index datatype shall be a finite, ordered datatype. [TCOS-LIS/D4 [B6]].

2.2.2.4 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 [43]]

2.2.2.5 attempted domain: The MD to which a tracing domain attempts but fails to transfer a communicate or report.

2.2.2.6 attempted MTA: The MTA to which a tracing MTA attempts but fails to transfer a communicate or report.

2.2.2.7 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 [43]]

2.2.2.8 Boolean: A datatype with two values, named “TRUE” and “FALSE”, and a set of logical operations: NOT, OR, AND, etc. [TCOS-LIS/D4 [B6]].

2.2.2.9 choice: A datatype constructed from a sequence of base datatypes, each associated with a name. A choice value contains, for exactly one name, a value of the corresponding datatype. [TCOS-LIS/D4 [B6]].

2.2.2.10 class: A category into which objects are placed on the basis of both their purpose and their internal structure [IEEE Std 1224-1993 [43]]

2.2.2.11 client instance: A manifestation of the client that shares the input and output queues of the client with other instances.

2.2.2.12 client: Software that uses an interface. [IEEE Std 1224-1993 [43]]

2.2.2.13 concrete: A class, instances of which are permitted either by direct instantiation or the instantiation of its concrete subclasses. [IEEE Std 1224-1993 [43]]

2.2.2.14 datatype: A collection of distinguished values, together with a collection of characterizing operations on those values. [IEEE Std 1224-1993 [43]]

2.2.2.15 delivery queue: One of two alternative databases that the service uses to convey objects to the client of the MA interface.

2.2.2.16 event flag: A Boolean associated with a session and maintained by the service that is used to signal the arrival of objects in the delivery, retrieval, or input queue.

2.2.2.17 feature: A negotiable aspect of an interface.

2.2.2.18 functional unit: A group of related functions.

2.2.2.19 generic interface: A version of an interface that is independent of any particular programming language.

2.2.2.20 input queue: The database that the service uses to convey objects to the client of the MT interface.

2.2.2.21 interface: Either the MA interface or the MT interface without distinction, or one of the two in particular.

2.2.2.22 list: A datatype constructed from a base datatype. A list value contains a sequence of zero or more values of the base datatype.

Applying an operation to all members of a list may be supported through either of two programming paradigms. In the first, the sequencing control is provided by the application; in the second, it is provided by the implementation. [TCOS/LIS/D4 [B6]].

2.2.2.23 local environment: The environment of the local MTA.

2.2.2.24 local MD: The MD of which the local MTA is a part.

2.2.2.25 local MTA: The MTA that comprises the client and the service in the context of the MT interface.

2.2.2.26 MA interface: The X.400 Application API.

2.2.2.27 mail system gateway: Software that uses the MT interface (the client).

2.2.2.28 MOTIS: The set of ISO standards for Message-Oriented Text Interchange Systems. Defined in references [8] - [14].

2.2.2.29 MT interface: The X.400 Gateway API.

2.2.2.30 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 [43]]

2.2.2.31 neighboring MTA: An MTA with which the service is prepared to establish OSI connections in the context of the MT interface.

2.2.2.32 network address attributes: The **ISDN-Number**, **ISDN-Subaddress**, **Presentation-Address**, and **X.121-Address** attributes specific to the **OR-Address** class.

2.2.2.33 object: Any of the complex information objects created, examined, modified, or destroyed by means of the OM interface. [IEEE Std 1224-1993 [43]]

2.2.2.34 OM interface: The API to OSI Object Management, IEEE Std 1224-1993 [43].

2.2.2.35 opaque: A datatype with no order or other operations defined.

An opaque datatype may have associated names that identify distinguished values. [TCOS-LIS/D4 [B6]].

2.2.2.36 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 [43]]

2.2.2.37 organizational unit name attributes: The **Organizational-Unit-Name-1**, **Organizational-Unit-Name-2**, **Organizational-Unit-Name-3**, and **Organizational-Unit-Name-4** attributes specific to the **OR-Address** class.

2.2.2.38 originating domain: The MD at which a communicate or report originated.

2.2.2.39 output queue: The database that the client of the MT interface uses to convey objects to the service.

2.2.2.40 package: A group of related classes. [IEEE Std 1224-1993 [43]]

2.2.2.41 personal name attributes: The **Given-Name**, **Initials**, **Surname**, and **Generation** attributes specific to the **OR-Address** class.

2.2.2.42 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 [43]]

2.2.2.43 record: A datatype constructed from a sequence of base datatypes, each associated with a name. A record value contains, for each name, a value of the corresponding base datatype. [TCOS-LIS/D4 [B6]].

2.2.2.44 reserved: An object in the delivery, retrieval, or input queue that the client can access without first removing it from that queue, but that no other client can access simultaneously.

2.2.2.45 retrieval queue: One of two alternative databases that the service uses to convey objects to the client of the MA interface.

2.2.2.46 service: Software that implements an interface.

2.2.2.47 service interface: The interface as realized, for the benefit of the client, by the service as a whole. [IEEE Std 1224-1993 [43]]

2.2.2.48 set: A datatype constructed from a base datatype; a value of a set datatype contains an unordered collection of values of the base datatype, each occurring only once.

The is-member operation returns a Boolean value that depends on whether the specified value is a member of the set. Applying an operation to all members of a set may be supported through either of two programming paradigms. In the first, the sequencing control is provided by the application; in the second, it is provided by the implementation. [TCOS-LIS/D4 [B6]].

2.2.2.49 source domain: The MD that supplies a piece of trace information.

2.2.2.50 State: An unordered, finite datatype. Each state value is identified by an associated name. [TCOS-LIS/D4 [B6]].

2.2.2.51 status datatype: An abstract datatype whose values may be bound to “control” values as well as “data” values. [IEEE Std 1224-1993 [43]]

2.2.2.52 structured: A postal O/R address that specifies the postal address of a user by means of several attributes. Its structure is prescribed in some detail.

2.2.2.53 subject domain: The MD that contains the MTA embodied by the client and service of the MT interface.

2.2.2.54 subject message: When used in reference to a communicate, the communicate, if it is a message; or any of the messages denoted by the communicate, if it is a probe.

2.2.2.55 submission queue: The database that the client of the MA interface uses to convey objects to the service.

2.2.2.56 test methods: The software, procedures, or other means specified to measure conformance to a specification. [IEEE Std 1224-1993 [43]]

2.2.2.57 tracing domain: The MD that produces an external trace entry.

2.2.2.58 tracing MTA: The MTA that produces an internal trace entry.

2.2.2.59 unstructured: A postal O/R address that specifies the postal address of a user in a single attribute. Its structure is left largely unspecified.

2.2.2.60 X.400: The set of CCITT Recommendations on message handling systems. This term covers both the X.400 (1984) and X.400 (1988) recommendations.

2.2.2.61 X.400 (1984): The set of CCITT Recommendations on message handling systems approved in 1984. Defined in references [34] - [41].

2.2.2.62 X.400 (1988): The set of CCITT Recommendations on message handling systems approved in 1988. Defined in references [8] - [14].

2.2.2.63 X.400 Application API: The interface that makes the functionality of the MTS accessible to an MS or a UA, or the functionality of a simple MS accessible to a UA.

2.2.2.64 X.400 Gateway API: The interface that divides an MTA into two software components, a mail system gateway and an X.400 gateway service.

2.2.2.65 X.400 gateway service: Software that implements the MT interface (the service).

2.2.3 Abbreviations

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

2.2.3.1 1984: X.400(1984).

2.2.3.2 1988: X.400 (1988).

2.2.3.3 ADMD: Administration Management Domain.

2.2.3.4 API: Application Program Interface.

2.2.3.5 ASN.1: Abstract Syntax Notation One.

2.2.3.6 AU: Access Unit

2.2.3.7 BER: Basic Encoding Rules

2.2.3.8 C: Conditional.

2.2.3.9 CA: Certification Authority.

2.2.3.10 CCITT: International Telegraph and Telephone Consultative Committee.

2.2.3.11 CDS: Command Document Start.

2.2.3.12 DCS: Digital Command Signal

2.2.3.13 DDA: Domain-Defined Attribute.

2.2.3.14 DL: Distribution List.

2.2.3.15 DR: Delivery Report.

2.2.3.16 DS: Directory System.

2.2.3.17 EDI: Electronic Data Interchange.

2.2.3.18 EIT: Encoded Information Type.

2.2.3.19 FIF: Facsimile Information Field.

2.2.3.20 FU: Functional Unit.

2.2.3.21 G3: Group 3 (facsimile).

2.2.3.22 G4: Group 4 (facsimile).

2.2.3.23 IA5: International Alphabet No. 5.

2.2.3.24 IM: Interpersonal Messaging.

2.2.3.25 IPM: Interpersonal Message.

2.2.3.26 IPN: Interpersonal Notification.

2.2.3.27 ISDN: Integrated Services Digital Network.

2.2.3.28 ITA2: International Telex Alphabet No. 2.

2.2.3.29 M: Mandatory.

2.2.3.30 MA: Message Access

2.2.3.31 MD: Management Domain.

2.2.3.32 MH: Message Handling.

- 2.2.3.33 MHS:** Message Handling System.
- 2.2.3.34 MOTIS:** Message-Oriented Text Interchange Systems.
- 2.2.3.35 MS:** Message Store.
- 2.2.3.36 MT:** Message Transfer.
- 2.2.3.37 MTA:** Message Transfer Agent.
- 2.2.3.38 MTS:** Message Transfer System.
- 2.2.3.39 NBP:** Nonbasic Parameter.
- 2.2.3.40 NDR:** Nondelivery Report.
- 2.2.3.41 NRN:** Nonreceipt Notification.
- 2.2.3.42 O/R:** Originator/Recipient.
- 2.2.3.43 OM:** Object Management.
- 2.2.3.44 OSI:** Open Systems Interconnection.
- 2.2.3.45 P1:** Message Transfer Protocol.
- 2.2.3.46 P2:** Interpersonal Messaging Protocol.
- 2.2.3.47 P3:** Message Submission and Delivery Protocol.
- 2.2.3.48 P7:** MS Access Protocol.
- 2.2.3.49 PAEK:** Public Asymmetric Encryption Key.
- 2.2.3.50 PDAU:** Physical Delivery Access Unit.
- 2.2.3.51 PDS:** Physical Delivery System.
- 2.2.3.52 POSIX:** Portable Operating System Interface.
- 2.2.3.53 PRMD:** Private Management Domain.
- 2.2.3.54 RD:** Recipient Descriptor.
- 2.2.3.55 RN:** Receipt Notification.
- 2.2.3.56 SAEK:** Secret Asymmetric Encryption Key.
- 2.2.3.57 SFD:** Simple Formattable Document.
- 2.2.3.58 SM:** Secure Messaging.
- 2.2.3.59 UA:** User Agent.

3. Overview

This section provides an overview of the interfaces and their specifications. It also overviews the facilities provided by the OSI Object Management API, and introduces the concept of features.

3.1 Purpose

This standard defines two application program interfaces (APIs) of architectural importance in the construction of message handling systems (MHS). It defines an X.400 Application API that makes the functionality of a message transfer system (MTS) accessible to a message store (MS) or user agent (UA), or the functionality of a simple MS

accessible to a UA. It also defines an X.400 Gateway API that divides a message transfer agent (MTA) into two software components, a mail system gateway and an X.400 gateway service.

The two interfaces support Message Access (MA), Message Transfer (MT), and Interpersonal Messaging (IM). MA is the conveyance of information objects between the MTS and one of its users, and thereby among its various users. MT is the conveyance of such objects between the MTAs that make up the MTS. IM is the MA and MT application in which the contents of messages are electronic memos. The interfaces address the operational requirements of these activities; they do not address their system management requirements, for example, those for security.

NOTE — Of the many applications of MA and MT, IM is singled out by this standard only because it is singled out by X.400 itself. (X.400 highlights IM for purely historical reasons.) The use of the interfaces is not limited to IM. Since they permit the contents of messages to be arbitrary binary data, the interfaces are germane to all applications, e.g., electronic data interchange (EDI).

Henceforth, the following terms are used consistently as indicated. The terms *MA interface* and *MT interface* denote the X.400 Application API and the X.400 Gateway API, respectively. The term *interface* denotes either interface without distinction, or one interface in particular. If the latter, which interface is meant will be clear from the context. The term *service* denotes software that implements an interface. A service that implements the MA interface realizes the MTS. A service that implements the MT interface realizes an X.400 gateway service. The term *client* denotes software that uses an interface. A client that uses the MA interface realizes a UA or an MS. A client that uses the MT interface realizes a mail system gateway.

NOTE — The MA interface does not provide the functionality of the MS Abstract Service of X.400 (its P7 protocol), although it can be viewed as providing a small portion of it. The functionality of P7 is outside the scope of this standard. The message storage and retrieval capabilities of a mail system are among its most important product-differentiating aspects. So that the interface does not effectively standardize those capabilities, their provision is made the responsibility of the client, not the service, and thus the design of those capabilities is left the responsibility of the client manufacturer.

3.2 Overview

The MA interface is high-level. It centers around three persistent databases that the service maintains even when it is out of contact with the client. The client places outbound messages and probes in a submission queue (the first database) for subsequent delivery or deliverability verification. The service places inbound messages and reports in a delivery queue (the second database) for subsequent delivery to the client. Alternatively, it places them in a retrieval queue (delivering them to the associated user in the process) for subsequent retrieval.

The MT interface also is high-level. It centers around two persistent databases that the service maintains even when it is out of contact with the client. The client places outbound messages, probes, and reports in the output queue (the first database) for subsequent action by the service. The service places inbound messages, probes, and reports in the input queue (the second database) for subsequent action by the client.

Both interfaces are explicit. They do not rely upon implicit communication channels between the client and the service. In particular, they neither presuppose nor exploit their use of a common file system or storage manager. They do assume, however, that the underlying operating system provides a means for signalling events (for efficient implementation of the `mx_wait` operations).

3.3 Object Management

The interfaces are designed to be used and implemented in conjunction with the 3o use and implementation of the general-purpose Object Management (OM) API defined in IEEE Std 1224-1993 [43]. Throughout this standard, the term *OM interface* denotes the OM API, the facilities of which are introduced briefly below.

The subclauses below introduce the various concepts that are used in Object Management, starting with the smallest.

3.3.1 Syntax

A *syntax* is the basis for the classification and representation of values in Object Management. Examples of syntaxes are Boolean, Integer, String(Octet), and Object.

Syntaxes are defined in the OM specification, and nowhere else, and they are themselves represented by integers.

3.3.2 Value

A *value* is a single datum, or piece of information. Each value belongs to exactly one syntax by which its representation is defined. A value may be as simple as a Boolean value (e.g., True), or as complicated as an entire OM object (e.g., a Message).

3.3.3 OM Attribute

An *OM attribute type* is an arbitrary category into which a specification places some values. OM attribute types are represented by integers, which are assigned in individual service specifications, and which are only meaningful within a particular package (see 3.3.6).

An *OM attribute* is an OM attribute type, together with an ordered sequence of 18 one or more values. OM attributes can occur only as parts of an OM object, and the OM attribute type and values are constrained by the OM class specification of that OM object (see 3.3.5). The OM attribute type can be thought of as the name of the OM attribute.

NOTE — There is no general representation for an OM attribute; however, individual language bindings may provide datatypes to allow an application to handle attributes in a convenient manner. (for example, the *OM_descriptor* data structure in the C language binding defined by IEEE Std 1224-1993 [43]).

3.3.4 OM Object

An *OM object* is a collection of OM attributes, the values of which can be accessed by means of operations. The particular OM attribute types that may occur in an OM object are determined by the OM class of the OM object (see 3.3.5), as are the constraints on those OM attributes. The OM class of an OM object is determined when the OM object is created and cannot be changed.

OM objects are represented in the interface by a handle or opaque pointer.

NOTE — The internal representation of an OM object is not specified; however, individual language bindings may provide datatypes to allow an application to handle objects in a convenient manner. (for example, the *OM_descriptor_list* data structure in the C language binding defined by IEEE Std 1224-1993 [43]).

3.3.5 OM Class

An *OM class* is a category of OM object, set out in a specification. It determines the OM attributes that may be present in the OM object and details the constraints on those OM attributes.

Each OM object belongs directly to exactly one OM class and is called an *instance* of that OM class.

The OM classes of OM objects form a tree; each OM class has exactly one immediate *superclass* (except for the OM class *Object*, which is the root of the tree), and each OM class may have an arbitrary number of *subclasses*. The tree structure is also known as the *OM class hierarchy*. The importance of the OM class hierarchy stems from the inheritance property that is discussed below.

Each OM class of OM object has a fixed list of OM attribute types, and every OM object that is an instance of the OM class has only these OM attributes (actually, some OM attributes may not be present in particular instances, as permitted by the constraints in the OM class specification). The list of OM attribute types that may appear in instances

of an OM class has two parts. Each OM class *inherits* all the OM attribute types that are permitted in its immediate superclass as legal OM attribute types. There is also a list of additional OM attribute types that are permitted in the OM class. Any subclasses of this OM class will inherit all of these OM attribute types from both lists.

Because of inheritance, an OM object is also said to be an instance of all its superclasses. It is required that the OM class constraints of each superclass are met, considering just those OM attribute types that are permitted in the superclass.

The OM class hierarchy and the list of OM attribute types for each OM class are determined solely by the interface specification and cannot be changed by a program.

The OM class specification may impose arbitrary constraints on the OM attributes. The most common of these are tabulated in the OM class specification and are marked with a * below. Frequently encountered cases include constraints to

- Restrict the syntaxes permitted for the values of an OM attribute (often to a single syntax)*
- Restrict the particular values to a subset of those permitted by the syntax
- Require exactly one value of the OM attribute (a *mandatory* OM attribute)*
- Require either zero or one value of the OM attribute (an *optional* OM attribute)*
- Permit multiple values, perhaps up to some limit known as the *value number constraint**
- Restrict the length of strings, up to a limit known as the *value length constraint**

NOTE — The constraint is expressed in terms of bits, octets, or characters according to the kind of string. The length of a string is indicated by the number of octets by which it is represented. The number of octets in a string with multibyte characters (e.g., T.61 Teletext) may thus exceed the value length constraint. (In a language binding, an array with more bytes will be needed to store it.)

The constraints may affect multiple OM attributes at once; for example, a rule that only one of several OM attributes may be present in any OM object.

Every OM object includes the OM class to which it belongs as the single value of the mandatory OM attribute type **Class**, which cannot be modified. The value of this OM attribute is an OSI Object Identifier, which is assigned to the OM class by the specification.

An *abstract class* is an OM class of which instances are forbidden. It may be defined as a superclass in order to share OM attributes between OM classes, or simply to ensure that the OM class hierarchy is convenient for the interface definition.

3.3.6 Package

A *package* is a set of OM classes that are grouped together by the specification because they are functionally related.

A package is identified by an OSI Object Identifier, which is assigned to the package by the specification. Thus the identity of each package is completely unique.

3.3.7 Package Closure

An OM class may be defined to have an OM attribute whose value is an OM object of an OM class that is defined in some other package. This is done to share definitions and to avoid duplication. For example, the Message Handling package defined in Section 6 defines an OM class called **OR-Name**. This OM class has an OM attribute whose value is an OM object of OM class **Name**, which is defined in the Directory Service package in the directory service API, IEEE Std 1224.2-1993 [44]. An OM class may also be a subclass of an OM class in another package. These relationships between packages lead to the concept of a package-closure.

A *package-closure* is 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. (A formal definition is given in IEEE Std 1224-1993 [43]).

3.3.8 Workspace

Details of the representation of OM objects and of the implementation of the operations that are used to manipulate them are not specified because they are not the concern of the application programmer. However, the programmer sometimes needs to be aware of which implementation is being used for a particular OM object.

The case of the OM class **Name** was mentioned in the previous subclause. This OM class is used in both the Message Transfer Service and in the Directory Service API, IEEE Std 1224.2-1993 [44]. If an application uses both services and the two services use different internal representations of OM objects (perhaps because they are supplied by different vendors), then it is necessary for the application to specify which implementation should create a **Name** OM object. This is done by means of a workspace.

A *workspace* is one or more package-closures, together with an implementation of the OM operations that support all the OM classes of OM objects in the package-closures.

The notion of a workspace also includes the storage used to represent OM objects and management of that storage. The interested reader can refer to IEEE Std 1224-1993 [43] for more details on how workspaces are implemented.

The application shall obtain a workspace that supports an OM class before it creates any OM objects of that OM class. The workspaces are returned by operations in the appropriate service. For example, `mt_open` returns a workspace that supports the Message Handling package, while another operation in another OSI service returns a workspace that supports another package.

Some implementations may support additional packages in a workspace. For example, vendors may provide OM classes for additional security classes in the Secure Messaging package. (The operation `mx_open` may be used to negotiate these packages into the workspace.) Another important case is where two or more services are supported by the same implementation. In this case, the workspaces returned by the Directory Service operation `ds_initialize` and the Message Transfer Service operation `mt_open` are likely to be the same one. The application need take no account of this, but may experience improved performance.

3.3.9 Use of Objects

OM objects are used to represent the data collections used in the interface, such as parts of a message or messaging operation results. Refer to Sections 5, 6, and 7 for the definition of these OM classes.

An important feature of the interface is that an instance of a subclass can be used wherever a particular OM class is needed. This means both that the application can supply a subclass and that the service can return a subclass. For example, the application can submit messages in any format that is defined as a subclass of the class **Submitted-Communique**, and the service returns all results of the `ma_submit` operation in any subclass of the **Submission-Results** class.

Because the service may return a subclass of the specified OM class, applications should always use the `om_instance` operation when checking the OM class of an OM object, rather than testing the value of the **Class** OM attribute.

When the application supplies a subclass of the specified OM class as an argument, the service will either recognize it as a vendor extension or will ignore all OM attribute types that are not permitted in the specified OM class.

The application can generally supply either a public object or a private object as an argument of the interface operations. There are exceptions such as the **Session** argument, which must be a private object in the interests of efficiency. The interface will always return private objects. The application can convert these into public objects by a call to `om_get` if required.

Note that public objects returned by `om_get` are read-only and shall not be modified in any way by the application.

3.4 Features

The interfaces enable the client and service to negotiate the use of the various defined features of the MA and MT interfaces. The *features* that this standard defines are the functional units (see below) and packages defined by them, and aspects of those functional units and packages identified as features. A *functional unit* (FU) is a group of related operations. Other standards or other versions of this standard may define (e.g., proprietarily) additional features, which may (but need not) themselves be FUs or packages.

The features defined by this standard are identified in Table 3.1. The first column of the table identifies the feature. The second column gives the symbolic name of an object identifier that denotes the feature. The ASN.1 object identifiers that denote each feature are given in Section 8..

The service makes a particular feature available to the client only if the client requests it. The client may request a set of features as a parameter to the `mx_open` operation of the MA or MT interface. The service provides the Basic Access FU and Basic Transfer FU respectively as default for the two interfaces.

The following takes place if the client attempts to use a feature that the service has not made available. If the client invokes an operation in the Basic Access or Basic Transfer FU, but that FU is unavailable, the behavior of the service is undefined. If the Basic Access FU is available and the client invokes a function in the Submission, Delivery, or Retrieval FU, but that FU is unavailable, the feature-unavailable error arises. If the Basic Transfer FU is available and the client invokes an operation in the Transfer In or Transfer Out FU, but that FU is unavailable, the feature-unavailable error arises. If the client supplies as an operation argument an object, one of whose subobjects is an instance of a class that is not in an available package, the no-such-class error arises.

The following feature pairs are mutually exclusive for a particular session between the client and the service: the Basic Access FU and the Basic Transfer FU, and the Delivery FU and the Retrieval FU.

Table 3.1—Features and Their Names

Feature	Name
Basic Access FU	basic-access
Submission FU	submission
Delivery FU	delivery
Retrieval FU	retrieval
Basic Transfer FU	basic-transfer
Transfer In FU	transfer-in
Transfer Out FU	transfer-out
IM 84 Package	im-84
IM 88 Package	im-88
MH 84 Package	mh-84
MH 88 Package	mh-88
SM 88 Package	sm-88
Multiple Delivery	multiple-delivery

NOTE — The delivery of a message or report to several local recipients as the result of a single invocation of the `ma_start_delivery` operation. In the absence of this feature, the service confines itself to one recipient for each invocation.

4. Message Handling Interfaces

This section defines the MA and MT interfaces. It specifies the operations that the service makes available to the client, the datatypes of which the arguments and results of the operations are data values, and the return codes that denote the outcomes (in particular, the exceptions) that the operations may report.

4.1 Datatypes

This section defines, and Table 4.1 lists, the datatypes of the MA and MT interfaces that are specific to MH. The interfaces also include the Boolean, Object, Object Identifier, Private Object, Return Code, String, and intermediate datatypes of the OM interface.

Table 4.1—Interface Datatypes Specific to MH

Data Type	Description
Boolean List	The result of feature negotiation for a session.
Feature List	The features to be negotiated for a session.
Interval	An interval of time measured in milliseconds.
Object Count	A number of objects.
Sequence Number	The sequence number of an object in a retrieval queue.

4.1.1 Boolean List

4.1.1.1 Synopsis

Datatype: mh_boolean_list_type
Definition: list[om_boolean_type]
Description: A list of boolean values, used to indicate activated features.

4.1.2 Feature List

4.1.2.1 Synopsis

Datatype: mh_feature_list_type
Definition: list[om_object_id_type]
Description: A list of features

4.1.3 Interval

4.1.3.1 Synopsis

Datatype: mh_interval_type
Definition: om_unsigned_integer_32_type
Description: A data type value of this datatype is the integer in the interval $[0, 2^{32})$ that denotes an interval of time measured in milliseconds.

4.1.4 Object Count

4.1.4.1 Synopsis

Datatype: mh_object_count_type
Definition: om_unsigned_integer_32_type
Description: A data type value of this datatype is the integer in the interval $[0, 2^{32})$ that denotes a number of objects.

4.1.5 Sequence Number

4.1.5.1 Synopsis

Datatype: mh_sequence_number_type
Definition: om_unsigned_integer_32_type
Description: A data type value of this datatype is the integer in the interval $[0, 2^{31})$ (*sic*) that denotes a message or report in a retrieval queue.

Sequence numbers are assigned in ascending order, but not necessarily consecutively. The sequence number of an object never changes, and no sequence number denotes two different objects, even at different times.

4.2 Access Operations

This section defines, and Table 4.2 lists, the operations of the MA interface.

Table 4.2—MA Interface Operations

Operation	Description
Cancel Submission	Cancel the deferred delivery of a submitted message.
Close	Terminate an MA session.
Finish Delivery	Conclude the delivery in progress in a session.
Finish Retrieval	Conclude the retrieval in progress in a session.
Open	Establish an MA session.
Size	Determine the size of the delivery or retrieval queue.
Start Delivery	Begin the delivery of a message or a report.
Start Retrieval	Begin the retrieval of a message or a report.
Submit	Submit a communicate.
Wait	Return when an object is available for delivery or retrieval.

As indicated in Table 4.2, the MA interface is comprised of a number of operations whose purpose and range of capabilities are summarized as follows:

Cancel Submission	This operation cancels a submitted message, with the deferred delivery option.
Close	This operation terminates an MA session between the client and the service. If the delivery or retrieval of a message or a report is in progress, the service first unsuccessfully finishes that delivery or retrieval.
Finish Delivery	This operation concludes the delivery in progress in a session. The client supplies delivery confirmations, as required, for users to which the object, a message, was delivered. It also indicates to which users the object, either a message or a report, is undeliverable.
Finish Retrieval	This operation concludes the retrieval in progress in a session. The client indicates whether the service is to remove the retrieved message or report from the retrieval queue or leave it there.
Open	This operation establishes an MA session between the client and the service, and makes the Basic Access FU and the OM Package initially available in that session. The client may also specify the other features required for the session. The client specifies either its own name or the O/R address of a local user. The session provides MTS access to the local user at the specified address, if the name of the client is specified, or to a group of local users statically associated with the client name, if the O/R address of a local user is specified.
Size	This operation determines the number of messages and reports in the delivery or retrieval queue to which a session provides access.
Start Delivery	This operation begins the delivery of a message or a report to one or more of the users associated with a session. If no messages or reports await delivery, the operation reports an exception.
Start Retrieval	This operation begins the retrieval of a message or a report from the retrieval queue to which a session provides access. If no messages or reports await retrieval, the operation reports an exception.
Submit	This operation submits a communicate (see 6.2.5) by adding it to the submission queue to which a session provides access. The operation first verifies the integrity of the communicate.

Wait

This operation returns when a message or a report is available for delivery or retrieval in the delivery or retrieval queue to which a session provides access, or when a period of time elapses, whichever occurs first.

The operations are grouped into four FUs (one basic and one each for submission, delivery, and retrieval) as indicated in Table 4.3 below. (As stated previously, the Delivery and Retrieval FUs are mutually exclusive.)

Table 4.3—MA Interface Operational Units

Basic Access	Submission	Delivery	Retrieval
Open	Submit	Size	Size
Close	Cancel Submission	Start Delivery	Start Retrieval
OM API		Finish Delivery	Finish Retrieval
		Wait	Wait

NOTE — The OM API is defined in IEEE Std 1224-1993 [43].

The intent of the interface definition is that each operation is atomic, i.e., that it either carries out its assigned task in full and reports success or fails to carry out even a portion of the task and reports an exception. However, the service does not guarantee that a task will not occasionally be carried out in part but not in full.

NOTE — Making such a guarantee might be prohibitively expensive.

Whether a operation detects and reports each of the exceptions listed in the Errors clause of its specification is unspecified. If a operation detects two or more exceptions, which it reports is unspecified. If a operation reports an exception for which a return code is defined, however, it uses that (rather than another) return code to do so.

4.2.1 Cancel Submission

4.2.1.1 Synopsis

Operation: ma_cancel_submission
Status Datatype: om_return_code_type
Input: session:om_private_object_type
mts_identifier:om_object_type

4.2.1.2 Description

This operation cancels the deferred delivery of a message, without regard to the session in which it was submitted.

4.2.1.3 Arguments In

session An established MA session between the client and the service; an instance of the **Session** class.
mts_identifier The MTS identifier assigned to the message whose delivery is to be canceled; an instance of the **MTS-Identifier** class.

4.2.1.4 Results

status Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.1.5.

4.2.1.5 Errors

feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_message, no_such_representation, no_such_session, no_such_syntax, no_such_type, not_private, permanent_error, system_error, temporary_error, too_late, too_many_values, wrong_class, wrong_value_length, wrong_value_makeup, wrong_value_number, wrong_value_syntax, or wrong_value_type

4.2.2 Close

4.2.2.1 Synopsis

Operation: ma_close
Status Datatype: om_return_code_type
Input: session:om_private_object_type

4.2.2.2 Description

This operation terminates an MA session, as well as destroying the workspace, 8 between the client and the service. If the delivery or retrieval of a message or a report is in progress in the session, the service first unsuccessfully finishes that delivery or retrieval.

4.2.2.3 Arguments In

session An established MA session between the client and the service; an instance of the **Session** class

4.2.2.4 Results

status Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.2.5.

4.2.2.5 Errors

operation_interrupted, memory_insufficient, network_error, no_such_session, not_private, permanent_error, system_error, temporary_error, or wrong_class

4.2.3 Finish Delivery

4.2.3.1 Synopsis

Operation: ma_finish_delivery
Status Datatype: om_return_code_type
Input: session:om_private_object_type
 delivery_confirmations:om_object_type
 non_delivery_reports:om_object_type

4.2.3.2 Description

This operation concludes the delivery in progress in a session. The client should indicate to which users the object, either a message or a report, is undeliverable by supplying nondelivery reports. The client may optionally supply

delivery confirmations for users to which the object, a message, was delivered. If the client does not explicitly confirm delivery or nondelivery, for all of the recipients on a message, then the service assumes that the message has been delivered to the recipients not covered by a confirmation or nondelivery.

The client indicates to which users the message or report is temporarily, rather than permanently, undeliverable. The circumstances that cause temporary undeliverability are client-defined. However, whether the service will accept any circumstances as causes of temporary failure is service implementation defined. If the service does not support temporary failures, it treats them as permanent.

If the object is a message, the service issues delivery reports (DRs), as required, for the users to which it has been delivered; issues NDRs, as required, for the users to which it is permanently undeliverable; and returns the message to the delivery queue for the users to which it is temporarily undeliverable. In the first case, the service considers that it has transferred responsibility for the message to the client.

If the object is a report, the service returns the report to the delivery queue for the users to which it is temporarily undeliverable. For the users to which the report has been delivered, the service considers that it has transferred responsibility for the report to the client.

4.2.3.3 Arguments In

<i>session</i>	An established MA session between the client and the service; an instance of the Session class. A delivery shall be in progress.
<i>delivery_confirmations</i>	One or more delivery confirmations, as required, for users to which the object, a message, was delivered; an instance of the Local-Delivery-Confirmations class.
<i>non_delivery_reports</i>	This argument may be omitted. Indicates the one or more users to which the object cannot be delivered; an instance of the Local-NDR class. This argument is omitted if there are no such users.

4.2.3.4 Results

status	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.3.5.
--------	---

4.2.3.5 Errors

feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_object, no_such_representation, no_such_session, no_such_syntax, no_such_type, not_private, permanent_error, session_not_busy, system_error, temporary_error, too_many_values, wrong_class, wrong_value_length, wrong_value_makeup, wrong_value_number, wrong_value_syntax, or wrong_value_type

4.2.4 Finish Retrieval

4.2.4.1 Synopsis

Operation:	ma_finish_retrieval
Status Datatype:	om_return_code_type
Input:	session:om_private_object_type remove:om_boolean_type

4.2.4.2 Description

This operation concludes the retrieval in progress in a session. The client indicates whether the service is to remove the message or report from the retrieval queue, or leave it there. In the former case, the service considers that it has transferred responsibility for the object from the service to the client. In the latter case, the service makes the objects inaccessible (i.e., the object handles are made invalid); the associated communicate or report, however, can be obtained again in a subsequent `ma_start_retrieval` operation invocation.

4.2.4.3 Arguments In

<i>session</i>	An established MA session between the client and the service; an instance of the Session class. A retrieval shall be in progress.
<i>remove</i>	Whether the service is to remove the retrieved message or report from the retrieval queue, rather than leave it there.

4.2.4.4 Results

<i>status</i>	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.4.5.
---------------	---

4.2.4.5 Errors

`feature_unavailable`, `operation_interrupted`, `memory_insufficient`, `network_error`, `no_such_object`, `no_such_session`, `not_private`, `permanent_error`, `session_not_busy`, `system_error`, `temporary_error`, or `wrong_class`

4.2.5 Open

4.2.5.1 Synopsis

Operation:	<code>ma_open</code>
Status Datatype:	<code>om_return_code_type</code>
Input:	<code>usser_address:om_public_object_type</code> <code>client_name:om_string_type</code> <code>feature_list:om_feature_list_type</code>
Output:	<code>session:om_private_object_type</code> <code>workspace:om_work space_type</code> <code>activated:om_boolean_list_type</code>

4.2.5.2 Description

This operation establishes an MA session between the client and the service, and it makes the Basic Access FU and the OM Package initially available in that session. The client may also specify the other features required for the session.

The client specifies either its own name or the O/R address of a local user. The session provides MTS access to the local user at the specified address, if the name of the client is specified, or to a group of local users statically associated with the client name, if the O/R address of a local user is specified. The Retrieval FU can be requested only if a single user is designated, but that user may be designated in either of these two ways.

The client always designates a particular user in the same way. The choice between O/R address and client name is made by means outside the scope of this standard. How users are associated with a client name also is outside the scope of this standard. The maximum number of users in a group is implementation defined (and may be one).

Opening an MA session also creates a workspace. A workspace contains objects returned as a result of operations invoked within that session. The workspace is used as an argument in the `om_create` and `om_copy` operations.

The maximum number of sessions that may exist simultaneously is implementation defined and may vary with time.

4.2.5.3 Arguments In

<i>user_address</i>	An explicit identification of the local user to which the session is to provide MTS access; an instance of the OR-Address class. If this argument is absent, the <i>client_name</i> argument shall be present.
<i>client_name</i>	The name by which the service knows the client, interpreted as a value whose syntax is String (IA5). It implicitly identifies one or more local users to which the session is to provide MTS access. If this argument is absent, the <i>user_address</i> argument shall be present.
<i>feature_list</i>	An ordered sequence of features, each represented by an object identifier.

4.2.5.4 Arguments Out

<i>session</i>	The established MA session between the client and the service; an instance of the Session class. The service prevents the client from modifying this object subsequently. This result is present if, and only if, the Return Code result is success.
<i>workspace</i>	The workspace that will contain all objects returned as a result of the operations invoked in the session.
<i>activated</i>	If successful, an ordered sequence of Boolean values with the same number of elements as the <i>feature-list</i> argument. If true, each value indicates that the corresponding feature is now part of the interface. If false, each value indicates that the corresponding feature is not available.

4.2.5.5 Results

<i>status</i>	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.5.6.
---------------	---

4.2.5.6 Errors

`feature_conflicts`, `feature_unavailable`, `operation_interrupted`, `memory_insufficient`, `network_error`, `no_such_client`, `no_such_representation`, `no_such_syntax`, `no_such_type`, `no_such_user`, `permanent_error`, `system_error`, `temporary_error`, `too_many_sessions`, `too_many_values`, `wrong_class`, `wrong_value_length`, `wrong_value_makeup`, `wrong_value_number`, `wrong_value_syntax`, or `wrong_value_type`

4.2.6 Size

4.2.6.1 Synopsis

Operation:	<i>ma_size</i>
Status Datatype:	<i>om_return_code_type</i>
Input:	<i>session:om_private_object_type</i>
Output:	<i>number:mh_object_count_type</i>

4.2.6.2 Description

This operation determines the number of messages and reports in the delivery or retrieval queue to which a session provides access.

4.2.6.3 Arguments In

<i>session</i>	An established MA session between the client and the service; an instance of the Session class.
----------------	--

4.2.6.4 Arguments Out

<i>number</i>	The number of objects in the delivery or retrieval queue. However, if that number exceeds $2^{16} - 1$, $2^{16} - 1$ is returned. This result is present if, and only if, the Return Code result is success.
---------------	---

4.2.6.5 Results

<i>status</i>	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.6.6.
---------------	---

4.2.6.6 Errors

feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_session, not_private, permanent_error, system_error, temporary_error, or wrong_class

4.2.7 Start Delivery

4.2.7.1 Synopsis

Operation:	ma_start_delivery
Status Datatype:	om_return_code_type
Input:	session:om_private_object_type
Output:	object:om_private_object_type

4.2.7.2 Description

This operation begins the delivery of a message or a report to one or more of the users associated with a session (see the definition of the **ma_open** operation). If no objects await delivery, the operation reports an exception. The client shall finish the delivery of one object before it starts the delivery of another in the same session. The delivery of a particular object cannot simultaneously be in progress in two sessions.

Whether the service begins the delivery of an object addressed to several users in one operation invocation, or in one invocation per user, is dependent on the multiple-delivery feature available for the session.

Which qualifying object in the delivery queue (if there are several such objects) the service selects for delivery is implementation defined.

NOTES:

- 1 — The invocation of this operation initiates, but does not consummate, delivery; that is, it does not transfer responsibility for the object from the service to the client. This is accomplished by means of the **ma_finish_delivery** operation.

- 2 — Although the `ma_wait` or `ma_size` operation indicates that the delivery queue contained an object immediately prior to invocation of this operation, this does not guarantee the success of this operation. For example, another process might have begun the delivery of the object.

4.2.7.3 Arguments In

session An established MA session between the client and the service; an instance of the **Session** class.

4.2.7.4 Arguments Out

object The object whose delivery is started; an instance of the **Delivered-Message** or the **Delivered-Report** class. If the former, the object includes one envelope for each of one or more of the users associated with the session. The service prevents the client from modifying this object subsequently. This result is present if, and only if, the Return Code result is success.

4.2.7.5 Results

status Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.7.6.

4.2.7.6 Errors

`bad_message`, `feature_unavailable`, `operation_interrupted`, `memory_insufficient`, `network_error`, `no_such_session`, `not_private`, `permanent_error`, `queue_empty`, `session_busy`, `system_error`, `temporary_error`, or `wrong_class`

4.2.8 Start Retrieval

4.2.8.1 Synopsis

Operation:	<code>ma_start_retrieval</code>
Status Datatype:	<code>om_return_code_type</code>
Input:	<code>session:om_private_object_type</code> <code>minimum_sequence_number:mh_sequence_number_type</code>
Output:	<code>selected_sequence_number:mh_sequence_number_type</code> <code>object:om_private_object_type</code>

4.2.8.2 Description

This operation begins the retrieval of a message or a report from the retrieval queue to which a session provides access. The client shall finish the retrieval of one object before it starts the retrieval of another in the same session. The retrieval of a particular object cannot simultaneously be in progress in two sessions.

The service selects for retrieval the object whose sequence number is nearest to but no less than a sequence number specified by the client. If no object has a sequence number greater than or equal to that specified, the operation reports an exception.

NOTES:

- 1 — The invocation of this operation initiates, but does not consummate, retrieval. That is, it does not transfer responsibility for the object from the service to the client. That is accomplished, if desired, by means of the `ma_finish_retrieval` operation.

- 2 — Although the `ma_wait` or `ma_size` operation indicates that the retrieval queue contained an object immediately prior to invocation of this operation, this does not guarantee the success of this operation. For example, another process might have begun the retrieval of the object.

4.2.8.3 Arguments In

<i>session</i>	An established MA session between the client and the service; an instance of the Session class.
<i>minimum_sequence_number</i>	The sequence number of the first message or report to be considered for retrieval.

4.2.8.4 Arguments Out

<i>selected_sequence_number</i>	The sequence number of the message or report selected for retrieval. This result is present if, and only if, the Return Code result is success.
<i>object</i>	The object whose retrieval is started; an instance of the Delivered-Message or the Delivered-Report class. If the former, the object includes a single envelope for the user associated with the session. The service prevents the client from modifying this object subsequently. This result is present if, and only if, the Return Code result is success.

4.2.8.5 Results

<i>status</i>	Whether the operation succeeded and, if not, why. It may be success or one of the values listed under Errors below.
---------------	---

4.2.8.6 Errors

feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_session, not_private, permanent_error, queue_empty, session_busy, system_error, temporary_error, or wrong_class

4.2.9 Submit

4.2.9.1 Synopsis

Operation:	<code>ma_submit</code>
Status Datatype:	<code>om_return_code_type</code>
Input:	<code>session:om_private_object_type</code> <code>communique:om_object_type</code>
Output:	<code>submission_results:om_private_object_type</code>

4.2.9.2 Description

This operation submits a communique by adding it to the submission queue to which a session provides access. This transfers responsibility for the communique from the client to the service. The operation first verifies the integrity of the communique.

4.2.9.3 Arguments In

<i>session</i>	An established MA session between the client and the service; an instance of the Session class.
<i>communique</i>	The object to be submitted; an instance of the Submitted-Communique class. Its purported originator shall be among the users associated with the session. The communique is made inaccessible if it is a private object.

4.2.9.4 Arguments Out

submission_results The results of the submission; an instance of the **Submission-Results** class. This result is present if, and only if, the Return Code result is success.

4.2.9.5 Results

status Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.2.9.6.

4.2.9.6 Errors

feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_class, no_such_object, no_such_representation, no_such_session, no_such_syntax, no_such_type, not_private, originator_improper, permanent_error, system_error, temporary_error, too_many_values, unsupported_critical_function, wrong_class, wrong_value_length, wrong_value_makeup, wrong_value_number, wrong_value_syntax, or wrong_value_type

4.2.10 Wait

4.2.10.1 Synopsis

Operation: ma_wait
Status Datatype: om_return_code_type
Input: session:om_private_object_type
interval:mh_interval_type
minimum_sequence_number:MH_sequence_number_type
Output: available:om_boolean_type

4.2.10.2 Description

This operation returns either when a message or a report is available for delivery or retrieval in the delivery or retrieval queue to which a session provides access, or when a period of time elapses, whichever occurs first. For retrieval queues, a message or a report is considered to be available when its sequence number is not less than a sequence number specified by the client.

For delivery queues:

- If there is at least one unreserved object on the queue to which the session provides access, the call returns immediately and flags available.
- If there are no unreserved objects on the queue and the specified interval is zero, the call returns immediately and flags unavailable.
- If there are no unreserved objects on the queue and the specified interval is greater than zero, then the call returns after the specified interval has elapsed or when the service places a new object on the queue, whichever occurs first. In the former case, availability is false, in the latter, true.
- If there are multiple sessions waiting on a single queue when an object arrives on the queue, to which and to how many the service returns early is implementation defined.

For retrieval queues:

- If there is at least one unreserved object with a sequence number greater than or equal to a sequence number specified by the client, and if this object is on the queue to which the session provides access, then the call returns immediately and flags available.

- If all the unreserved objects on the queue have sequence numbers strictly less than a sequence number specified by the client, and if the specified interval is zero, then the call returns immediately and flags unavailable.
- If all the unreserved objects on the queue have sequence numbers strictly less than a sequence number specified by the client and the specified interval is greater than zero, then the call returns after the specified interval has elapsed or when the service places a new object on the queue having a sequence number greater than or equal to the sequence number specified by the client, whichever occurs first. In the former case, availability is false, in the latter, true.
- If there are multiple sessions waiting on a single queue when an object arrives on the queue, to which and to how many the service returns early is implementation defined.

NOTE — This operation is designed to be easily implemented using the event signalling primitives of many operating systems, including the primitives whose inclusion in POSIX is currently under discussion within the IEEE.

4.2.10.3 Arguments In

<i>session</i>	An established MA session between the client and the service; an instance of the Session class.
<i>interval</i>	The maximum length of time that the service is to block the client before returning.
<i>minimum_sequence_number</i>	The sequence number, less than which objects on a retrieval queue are considered unavailable. This argument is absent for delivery queues.

4.2.10.4 Arguments Out

<i>available</i>	A Boolean that is true if there was an available object on the queue when the operation was called or if the call returned early when an object was placed on the queue.
------------------	--

4.2.10.5 Results

<i>status</i>	Whether the operation succeeded and, if not, why. It may be success or one of the values listed under Errors below.
---------------	---

4.2.10.6 Errors

feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_session, not_private, permanent_error, system_error, temporary_error, or wrong_class

4.3 Transfer Operations

This section defines, and Table 4.4 lists, the operations of the MT interface.

Table 4.4—MT Interface Operations

Operation	Description
Close	Terminate an MT session.
Finish Transfer In	Conclude one of the transfers in that is in progress in a session.
Open	Establish an MT session.
Size	Determine the size of the input queue.
Start Transfer In	Begin the transfer in of a communicate or a report.
Transfer Out	Add a communicate or report to the output queue.
Wait	Return when an object is available for transfer in.

As indicated in the table, the MT interface is comprised of a number of operations whose purpose and range of capabilities are summarized as follows:

Close	This operation terminates an MT session between the client and the service. If the transfer in of a communicate or a report is in progress, the service first unsuccessfully finishes that transfer in.
Finish Transfer In	This operation concludes one or all of the transfers in that are in progress in a session. The client indicates whether the service is to remove the objects from the input queue or leave them there.
Open	This operation establishes an MT session between the client and the service and makes the Basic Transfer FU and the OM Package initially available in that session. The client may also specify the other features required for the session. The client specifies its name and instance name.
Size	This operation determines the number of reserved objects and unreserved objects in the input queue to which a session provides access. Each object is a communicate or a report.
Start Transfer In	This operation begins the transfer in of a communicate or a report by reserving an unreserved object in the input queue. If no objects await transfer in, the operation reports an exception.
Transfer Out	This operation adds a communicate or a report to the output queue to which a session provides access. The operation first verifies the integrity of the object.
Wait	This operation returns when a communicate or a report is available for transfer in the input queue to which a session provides access, or when a period of time elapses, whichever occurs first.

The operations are grouped into three FUs (one basic and one each for transfer in and transfer out) as indicated in Table 4.5.

Table 4.5—MT Interface Operational Units

Basic Transfer	Transfer Out	Transfer In
Open	Transfer Out	Size
Close		Start Transfer In
OM API		Finish Transfer In
		Wait

NOTE — The OM API is defined in IEEE Std 1224-1993 [43].

The intent of the interface definition is that each operation is atomic, i.e., that it either carries out its assigned task in full and reports success or fails to carry out even a portion of the task and reports an exception. However, the service does not guarantee that a task will not occasionally be carried out in part but not in full.

NOTE — Making such a guarantee might be prohibitively expensive.

Whether a operation detects and reports each of the exceptions listed in the Errors clause of its specification is unspecified. If a operation detects two or more exceptions, which it reports is unspecified. If a operation reports an exception for which a return code is defined, however, it uses that (rather than another) return code to do so.

4.3.1 Close

4.3.1.1 Synopsis

Operation:	<code>mt_close</code>
Status Datatype:	<code>om_return_code_type</code>
Input:	<code>session: om_object_type</code>

4.3.1.2 Description

This operation terminates an MT session, as well as destroys the workspace, between the client and the service. If the transfer in of one or more communicates or reports are in progress in the session, the service first unsuccessfully finishes those transfers in.

4.3.1.3 Arguments In

<i>session</i>	An established MT session between the client and the service; an instance of the Session class.
----------------	--

4.3.1.4 Results

<i>status</i>	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.3.1.5.
---------------	---

4.3.1.5 Errors

`operation_interrupted`, `memory_insufficient`, `network_error`, `no_such_session`, `not_private`, `permanent_error`, `system_error`, `temporary_error`, or `wrong_class`

4.3.2 Finish Transfer In

4.3.2.1 Synopsis

Operation:	<code>mt_finish_transfer_in</code>
Status Datatype:	<code>om_return_code_type</code>
Input:	<code>session: om_private_object_type</code> <code>object: om_private_object_type</code> <code>all: om_boolean_type</code> <code>remove: om_boolean_type</code>

4.3.2.2 Description

This operation concludes one or all of the transfers in progress in a session. The client indicates whether the service is to remove the objects from the input queue or leave them there. In the former case, the service considers that it has

transferred responsibility for the objects from the service to the client. In the latter case, the service marks the objects unreserved and makes them inaccessible (i.e., the object handles are made invalid); the associated *communicate* or *report*, however, can be obtained again in a subsequent *mt_start_transfer_in* operation invocation.

The *om_delete* operation also unreserves the object to which it is applied.

NOTES:

- 1 — The client should invoke this operation, with the *remove* argument true, after it successfully processes an object in the input queue. If the object is a message or a report, such processing involves delivering or internally transferring the object to the indicated users of the mail system to which the client is a gateway. If the object is a probe, such processing involves establishing the deliverability to those users of a message of the variety the probe describes.
- 2 — An exceptional condition may prevent the client from successfully processing the object. If it judges the exception (e.g., the invalidity of an O/R address) to be permanent, the client should issue an NDR and permanently terminate the transfer in that is using the present operation with the *remove* argument true. (However, X.400 does not permit an undeliverable report itself to provoke an NDR, but rather requires that the report simply be discarded.) If the client judges the exception (e.g., a network failure) to be temporary, the client should temporarily terminate the transfer in that is using the present operation with the *remove* argument false and later reinitiate it using the *mt_start_transfer_in* operation.

4.3.2.3 Arguments In

<i>session</i>	An established MT session between the client and the service; an instance of the Session class.
<i>object</i>	The particular object whose transfer in is to be concluded; an instance of the Communicate or the Report class. The object shall be (currently) reserved. The object is made inaccessible. This argument is present if, and only if, the <i>all</i> argument is false.
<i>all</i>	Whether the transfer in of all reserved objects, rather than one particular reserved object, is to be concluded.
<i>remove</i>	Whether the service is to remove the object or objects from the input queue, rather than leave them there.

4.3.2.4 Results

<i>status</i>	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.3.2.5.
---------------	---

4.3.2.5 Errors

feature_unavailable, *operation_interrupted*, *memory_insufficient*, *network_error*, *no_such_object*, *no_such_session*, *not_private*, *not_reserved*, *permanent_error*, *session_not_busy*, *system_error*, *temporary_error*, or *wrong_class*

4.3.3 Open

4.3.3.1 Synopsis

Operation:	<i>mt_open</i>
Status Datatype:	<i>om_return_code_type</i>
Input:	<i>client_name</i> : <i>om_string_type</i> <i>client_instance_name</i> : <i>om_string_type</i> <i>feature_list</i> : <i>mh__feature_list_type</i>
Output:	<i>activated</i> : <i>om_boolean_list_type</i> <i>session</i> : <i>om_private_object_type</i>

workspace: om_workspace_type

4.3.3.2 Description

This operation establishes an MT session between the client and the service, and it makes the Basic Transfer FU and the OM Package initially available in that session. The client may also specify the other features required for the session.

The client specifies its own name and its instance name. If the service has several clients, the name of the client determines the input and output queues to which the session provides access. All *client instances* share the input and output queues of the client. If the client has several instances, the service may use the instance name, e.g., in log entries it makes for purposes of system management.

Opening an MT session also creates a workspace. A workspace contains objects returned as a result of operations invoked within that session. The workspace is used as an argument in the *om_create* and *om_copy* operations.

The maximum number of sessions that may exist simultaneously is implementation defined and may vary with time.

4.3.3.3 Arguments In

<i>client_name</i>	The name by which the service knows the client, interpreted as a value whose syntax is String (IA5).
<i>client_instance_name</i>	The name by which the service knows the client instance, interpreted as a value whose syntax is String (IA5).
<i>feature_list</i>	An ordered sequence of features, each represented by an object identifier.

4.3.3.4 Arguments Out

<i>activated</i>	If successful, an ordered sequence of Boolean values with the same number of elements as the <i>feature_list</i> argument. If true, each value indicates that the corresponding feature is now part of the interface. If false, each value indicates that the corresponding feature is not available.
<i>session</i>	The established MT session between the client and the service; an instance of the Session class. The service prevents the client from modifying this object subsequently. This result is present if, and only if, the Return Code result is success.
<i>workspace</i>	The workspace that will contain all objects returned as a result of the operations invoked in the session.

4.3.3.5 Results

status	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.3.3.6.
--------	---

4.3.3.6 Errors

feature_conflicts, feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_client, no_such_client_instance, permanent_error, system_error, temporary_error, too_many_sessions, or wrong_class

4.3.4 Size

4.3.4.1 Synopsis

Operation:	mt_size
Status Datatype:	om_return_code_type
Input:	session: om_private_object_type
Output:	number: mh_object count type

4.3.4.2 Description

This operation determines the number of objects in the input queue to which a session provides access. Each object is a communicate or a report.

4.3.4.3 Arguments In

<i>session</i>	An established MT session between the client and the service; an instance of the Session class.
----------------	--

4.3.4.4 Arguments Out

<i>number</i>	The number of unreserved objects in the input queue. However, if that number exceeds $2^{16} - 1$, $2^{16} - 1$ is returned. This result is present if, and only if, the Return Code result is success.
---------------	--

4.3.4.5 Results

status	Whether the operation succeeded and, if not, why. It may be success or one of the values listed in 4.3.4.6.
--------	---

4.3.4.6 Errors

feature_unavailable, operation_interrupted, memory_insufficient, network_error, no_such_session, not_private, permanent_error, system_error, temporary_error, or wrong_class

4.3.5 Start Transfer In

4.3.5.1 Synopsis

Operation:	mt_start_transfer_in
Status Datatype:	om_return_code_type
Input:	session: om_private_object_type
Output:	object: om_private_object_type identifier: om_string_type

4.3.5.2 Description

This operation begins the transfer in of a communicate or a report by reserving an unreserved object in the input queue to which a session provides access. If no objects await transfer in, the operation reports an exception. The client need not finish the transfer in of one object before it starts the transfer in of another in the same session. The transfer in of a particular object, however, cannot simultaneously be in progress in two sessions.

Whether the service begins the transfer in of a message addressed to several users in one operation invocation or several (e.g., one per user or one per group of users) is implementation defined.