

# IEEE Standard for Information Technology—Test Methods for Measuring Conformance to X.400-Based Electronic Messaging C Language Interfaces—Binding for Application Program Interface (API)

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

Approved March 18, 1993  
**IEEE Standards Board**

Approved September 16, 1993  
**American National Standards Institute**

**Abstract:** This standard defines test methods that may be used to test conformance to IEEE Std 1327.1-1993, which is a C language binding for IEEE Std 1224.1-1993, IEEE Standard for Information Technology—X.400-Based Electronic Messaging—Application Program Interface (API) [Language Independent].

**Keywords:** API, application program interface, C language, language binding, test methods

---

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 1-55937-315-6

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

**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 1328.1-1993, IEEE Standard for Information Technology—Test Methods for Measuring Conformance to X.400-Based Electronic Messaging C Language Interfaces—Binding for Application Program Interface (API), but is included for information only.)

## Purpose

The purpose of this standard is to specify the requirements for test methods for measuring conformance to IEEE Std 1327.1-1993 {6}, which defines a C programming language binding to the X.400-Based Electronic Messaging Application Program Interface (API).

## Related Standards

IEEE Std 1327.1-1993 {6} is a C programming language binding for the programming language-independent specification contained in IEEE Std 1224.1-1993 {4}. The requirements for test methods for measuring conformance to IEEE Std 1224.1-1993 {4} are contained in IEEE Std 1326.1-1993 {5}. The requirements specified in IEEE Std 1326.1-1993 {5} apply to test methods for measuring conformance to IEEE Std 1327.1-1993 {6}, in addition to the requirements specified in this standard.

IEEE Std 1327.1-1993 {6} is intended to be used in conjunction with IEEE Std 1224-1993 {3}, which provides a general purpose API for the creation, examination, modification, and deletion of OSI information objects.

The test assertions in this standard were produced in accordance with IEEE Std 1003.3-1991 {2}.

## Overview

This standard contains a section corresponding to each section of IEEE Std 1327.1-1993 {6}. Each such section contains test assertions for that section of IEEE Std 1327.1-1993 {6}. A set of test methods for measuring conformance to IEEE Std 1327.1-1993 {6} is required to test these assertions in conformance with the requirements of IEEE Std 1003.3-1991 {2}.

## 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:<sup>1</sup>

- 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 IEEE Standards procedures.

Approved supplements are published separately and distributed with orders from the IEEE for this document until the full document is reprinted and such supplements 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 supplements have been published.

---

<sup>1</sup>A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903 USA; Telephone: +1 202 371-0101; FAX: +1 202 728-96124.

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 |

#### **P1224 Working Group**

|                        |                     |                      |
|------------------------|---------------------|----------------------|
| Curt Campbell          | Mark Karan          | Pierre Perrot        |
| Tony Cincotta          | Eva Kuipar          | Enzo Signore         |
| Chris Harding          | Kevin<br>Montgomery | Daniella<br>Sirocchi |
| Khiem Ho               | Et Owens            | Andrew<br>Twigger    |
| Petr JaneDCL<br>135 ek |                     | 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** (*EurOpen Organizational Representative*)

**Ralph Barker** (*Uniforum Organizational Representative*)

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

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

**Wallace S. Read**, *Chair*

**Donald C. Loughry**, *Vice Chair*

**Andrew G. Salem**, *Secretary*

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

\* Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

|                    |                                      |                    |
|--------------------|--------------------------------------|--------------------|
| Satish K. Aggarwal | Richard B. Engelman                  | David E. Soffrin   |
| James Beall        |                                      | Stanley I. Warshaw |
|                    | Rachel Auslander                     |                    |
|                    | <i>IEEE Standards Project Editor</i> |                    |

| CLAUSE                                                                 | PAGE |
|------------------------------------------------------------------------|------|
| 1. General .....                                                       | 1    |
| 1.1 Scope .....                                                        | 1    |
| 1.2 Normative References .....                                         | 1    |
| 1.3 Conformance .....                                                  | 2    |
| 2. Terminology and General Requirements .....                          | 2    |
| 2.1 Conventions .....                                                  | 2    |
| 2.2 Definitions .....                                                  | 3    |
| 2.3 General Terms .....                                                | 3    |
| 2.4 Abbreviations .....                                                | 4    |
| 3. Test Methods for Section General .....                              | 4    |
| 4. Test Methods for Section Terminology and General Requirements ..... | 4    |
| 4.1 Conventions .....                                                  | 4    |
| 4.2 Definitions .....                                                  | 4    |
| 4.3 General Terms .....                                                | 5    |
| 4.4 Abbreviations .....                                                | 5    |
| 5. Test Methods for Section Message Handling Interfaces .....          | 5    |
| 5.1 Introduction .....                                                 | 5    |
| 5.2 Datatypes .....                                                    | 5    |
| 5.3 Access Functions .....                                             | 6    |
| 5.4 Transfer Functions .....                                           | 10   |
| 5.5 Return Codes .....                                                 | 13   |
| 5.6 Header File .....                                                  | 13   |
| 6. Test Methods for Section Interpersonal Messaging Packages .....     | 13   |
| 7. Test Methods for Section Message Handling Packages .....            | 21   |
| 8. Test Methods for Section Secure Messaging Packages .....            | 49   |
| Annex A Bibliography (Informative) .....                               | 53   |

# IEEE Standard for Information Technology—Test Methods for Measuring Conformance to X.400-Based Electronic Messaging C Language Interfaces—Binding for Application Program Interface (API)

## 1. General

### 1.1 Scope

This standard defines requirements for test methods for measuring conformance to IEEE Std 1327.1-1993 {6}.

IEEE Std 1327.1-1993 {6} contains a programming language binding specification for IEEE Std 1224.1-1993 {4}, using the C programming language. IEEE Std 1326.1-1993 {5} contains language-independent requirements for test methods for measuring conformance to programming language binding specifications for IEEE Std 1224.1-1993 {4}, such as that contained in IEEE Std 1327.1-1993 {6}. This standard contains C language specific requirements for the test methods. Taken in conjunction with the requirements imposed by IEEE Std 1326.1-1993 {5}, they constitute the requirements that shall be satisfied by test methods used for measuring conformance to IEEE Std 1327.1-1993 {6}.

### 1.2 Normative References

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

{1} ISO/IEC 9899:1990,<sup>1</sup>*Programming Languages—C*.

---

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

{2} IEEE Std 1003.3-1991,<sup>2</sup>*IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX*.

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

{4} IEEE Std 1224.1-1993, *IEEE Standard for Information Technology—X.400-Based Electronic Messaging Application Program Interface (API) [Language Independent]*.

{5} 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]*.

{6} IEEE Std 1327.1-1993, *IEEE Standard for Information Technology—X.400-Based Electronic Messaging C Language Interfaces—Binding for Application Program Interface (API)*.

## 1.3 Conformance

A set of test methods that conforms to this standard shall conform to IEEE Std 1003.3-1991 {2}, with references to the “POSIX.n test method specification” being interpreted as references to this standard, and references to “the POSIX standard for which conformance is being measured” being interpreted as references to IEEE Std 1327.1-1993 {6}.

In addition to meeting the conformance criteria defined in IEEE Std 1003.3-1991 {2}, a set of test methods that conforms to this standard shall test all documentation assertions defined in this standard.

NOTE — Conformance to IEEE Std 1003.3-1991 {2} implies that the test methods will test all other assertions defined in this standard.

A set of test methods that conforms to this standard shall also conform to IEEE Std 1326.1-1993 {5}.

## 2. Terminology and General Requirements

### 2.1 Conventions

#### 2.1.1 General and Typographic Conventions

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

- A C language function, datatype, or code fragment appears in *italic* font. A function name is indicated by following parentheses, for example, *ma\_submit()*.
- A symbolic constant, other than an error name, is surrounded by braces, for example, {OMP\_O\_MH\_FE\_BASIC\_ACCESS}.
- A symbolic constant for an error name is surrounded by brackets, for example, [MH\_RC\_BAD\_MESSAGE].

#### 2.1.2 C Naming Conventions

How the identifier for an element of a C interface is derived from the name of the corresponding element of the generic interface depends upon the type of element as specified in Table 1. The generic name is prefixed with the character

<sup>2</sup>IEEE documents can be obtained from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ, 08855-1331, USA.

string in the second column of the table, alphabetic characters are converted to the case in the third column, and an underscore ( `_` ) is substituted for each hyphen ( `-` ) or space (  ).

The prefix “mhP” is reserved for use by implementors of the service. The prefixes “mhX” and “MHX” are reserved for the proprietary extension of the interface. In all other respects, such extension is outside the scope of this standard.

### 2.1.3 Language-Independent Conventions

The language-independent conventions defined in IEEE Std 1224.1-1993 {4} shall apply in this standard.

**Table 2-1—Derivation of C Identifiers**

| Element Type                  | Prefix                 | Case  |
|-------------------------------|------------------------|-------|
| Datatype                      | MH_                    | Lower |
| Data value                    | IM_, MH_, SM_          | Upper |
| Data value (Class) [See note] | IM_C_, MH_C_, SM_C_    | Upper |
| Data value (Class) [See note] | IM_C_, MH_C, SM_C_     | Upper |
| Data value (Value Length)     | IM_VL, _MH_VL_, SM_VL_ | Upper |
| Data value (Value Number)     | IM_VN, _MH_VN_, SM_VN_ | Upper |
| Data value component          | none                   | Lower |
| Function                      | ma_, mt_               | Lower |
| Function argument             | none                   | Lower |
| Function result               | none                   | Lower |

NOTE — The C identifiers denote only the Elements components of data values of the OM Siting datatype.

## 2.2 Definitions

### 2.2.1 Terminology

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

**2.2.1.1 may:** An indication of an optional feature [ISO/IEC 9945-1: 1990 {B1}]. With respect to implementations, the word *may* is to be interpreted as an optional feature that is not required in this standard, but can be provided. With respect to Strictly Conforming Applications, the word *may* means that the optional feature shall not be used.

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

**2.2.1.3 should:**

- 1) With respect to implementations, an indication of an implementation recommendation, but not a requirement.
- 2) With respect to applications, an indication of a recommended programming practice for applications and a requirement for Strictly Conforming Applications [ISO/IEC 9945-1: 1990 {B1}].

## 2.3 General Terms

The terms used in this standard are the same as those defined in the language independent API specification from which it is derived.

## 2.4 Abbreviations

The abbreviations used in this standard are the same as those defined in the language-independent API specification from which it is derived.

## 3. Test Methods for Section General

There are no test assertions for this section.

## 4. Test Methods for Section Terminology and General Requirements

### 4.1 Conventions

#### 4.1.1 General and Typographic Conventions

#### 4.1.2 C Naming Conventions

001(A) The identifier for an element is derived from the name of the corresponding element of the generic interface, depending on the type of the element as defined in the following table:

**Table 4-1—Derivation of C Identifiers**

| Element Type              | Prefix                 | Case  |
|---------------------------|------------------------|-------|
| Datatype                  | MH_                    | Lower |
| Data value                | IM_, MH_, SM_          | Upper |
| Data value (Class)        | IM_C_, MH_C_, SM_C_    | Upper |
| Data value (Class)        | IM_C_, MH_C_, SM_C_    | Upper |
| Data value (Value Length) | IM_VL_, MH_VL_, SM_VL_ | Upper |
| Data value (Value Number) | IM_VN_, MH_VN_, SM_VN_ | Upper |
| Data value component      | none                   | Lower |
| Function                  | ma_, mt_               | Lower |
| Function argument         | none                   | Lower |
| Function result           | none                   | Lower |

#### 4.1.3 Language-Independent Conventions

There are no test assertions for this subclause.

### 4.2 Definitions

There are no test assertions for this clause.

## 4.3 General Terms

There are no test assertions for this clause.

## 4.4 Abbreviations

There are no test assertions for this clause.

# 5. Test Methods for Section Message Handling Interfaces

## 5.1 Introduction

## 5.2 Datatypes

### 5.2.1 Feature

#### 5.2.1.1 Synopsis

R01 The structure, `MH_feature` is declared as follows:

```
typedef struct {
    OM_object_identifier    feature;
    OM_boolean              activated;
} MH_feature;
```

### 5.2.2 Interval

#### 5.2.2.1 Synopsis

R02 The type `MH_interval` is declared as follows:

```
typedef OM_uint32 MH_interval;
```

### 5.2.3 Object Count

#### 5.2.3.1 Synopsis

R03 The type `MH_object_count` is declared as follows:

```
typedef OM_uint32 MH_object_count;
```

### 5.2.4 Sequence Number

#### 5.2.4.1 Synopsis

R04 The type `MH_sequence_number` is declared as follows:

```
typedef OM_uint32 MH_sequence_number;
```

## 5.3 Access Functions

### 5.3.1 Cancel Submission

#### 5.3.1.1 Synopsis

R05 If the Submission FU is supported:  
When the header file <xmh.h> is included, then the function *ma\_cancel\_submission()* is declared as follows:

```
OM_return_code
ma_cancel_submission (
    OM_private_object    session,
    OM_object            mts_identifier
);
```

#### 5.3.1.2 Results

001(D) If the Submission FU is supported:  
When *ma\_cancel\_submission()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.4.7.1.)

### 5.3.2 Close

#### 5.3.2.1 Synopsis

R06 If the Basic Access FU is supported:  
When the header file <xmh.h> is included, then the function *ma\_close()* is declared as follows:

```
OM_return_code
ma_close (
    OM_private_object    session
);
```

#### 5.3.2.2 Results

002(D) If the Basic Access FU is supported:  
When *ma\_close()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code (See GA01 in 5.4.7.1).

### 5.3.3 Finish Delivery()

#### 5.3.3.1 Synopsis

R07 If the Delivery FU is supported:  
When the header file <xmh.h> is included, then the function *ma\_finish\_delivery()* is declared as follows:

```
OM_return_code
ma_finish_delivery (
    OM_private_object    session,
    OM_object            delivery_confirmations,
    OM_object            non_delivery_reports
);
```

### 5.3.3.2 Description

003(C) If the Delivery FU is supported:

When *ma\_finish\_delivery*(*delivery\_confirmations*,) is called and completes successfully, then the absence of *delivery\_confirmations* is signaled by the null pointer.

004(C) If the Delivery FU is supported:

When *ma\_finish\_delivery*(*non\_delivery\_reports*) is called and completes successfully, then the absence of *non\_delivery\_reports* is signaled by the null pointer.

### 5.3.3.3 Results

005(D) If the Delivery FU is supported:

When *ma\_finish\_delivery*() is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.4.7.1.)

## 5.3.4 Finish Retrieval()

### 5.3.4.1 Synopsis

R08 If the Retrieval FU is supported:

When the header file <xmh.h> is included, then the function *ma\_finish\_retrieval*() is declared as follows:

```
OM_return_code
ma_finish_retrieval (
    OM_private_object    session,
    OM_boolean           remove
);
```

### 5.3.4.2 Results

006(D) If the Retrieval FU is supported:

When *ma\_finish\_retrieval*() is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.4.7.1.)

## 5.3.5 Open()

### 5.3.5.1 Synopsis

R09 If the Basic Access FU is supported:

When the header file <xmh.h> is included, then the function *ma\_open*() is declared as follows:

```
OM_return_code
ma_open (
    OM_public_object    user_address,
    OM_string           client_name,
    MH_feature          feature_list [],
    OM_private_object   *session,
    OM_workspace        *workspace
);
```

### 5.3.5.2 Description

- 007(C) If the Basic Access FU is supported:  
When *ma\_open*(*user\_address*,,,) is called and completes successfully, then the absence of *user\_address* is signaled by the null pointer.
- 008(C) If the Basic Access FU is supported:  
When *ma\_open*(*client\_name*,,,) is called and completes successfully, then the absence of *client\_name* is signaled by a zero length string.
- 009(C) If the Basic Access FU is supported:  
When *ma\_open*(*feature\_list*,,) is called and completes successfully, then the *feature\_list* sequence is terminated by an object identifier having a length of zero and having any value for the data pointer.

### 5.3.5.3 Results

- 010(D) If the Basic Access FU is supported:  
When *ma\_open*() is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.4.7.1.)

### 5.3.6 Size()

#### 5.3.6.1 Synopsis

- R10 If the Delivery or Retrieval FU is supported:  
When the header file <xmh.h> is included, then the function *ma\_size*() is declared as follows:

```
OM_return_code
ma_size (
    OM_privat_object    session,
    MH_object_count     *number
);
```

#### 5.3.6.2 Results

- 011(D) If the Delivery or Retrieval FU is supported:  
When *ma\_size*() is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.4.7.1.)

### 5.3.7 Start Delivery()

#### 5.3.7.1 Synopsis

- R11 If the Delivery FU is supported:  
When the header file <xmh.h> is included, then the function *ma\_start\_delivery*() is declared as follows:

```
OM return code
ma_start_delivery (
    OM_private_object    session,
    OM_private_object    *object
);
```

### 5.3.7.2 Results

012(D) If the Delivery FU is supported:

When *ma\_start\_delivery()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

### 5.3.8 Start Retrieval()

#### 5.3.8.1 Synopsis

R12 If the Retrieval FU is supported:

When the header file <xmh.h> is included, then the function *ma\_start\_retrieval()* is declared as follows:

```
OM_return_code
ma_start_retrieval (
    OM_private_object      session,
    MH_sequence_number     minimum_sequence_number,
    MH_sequence_number     *selected_sequence_number,
    OM_private_object      *object
);
```

#### 5.3.8.2 Results

013(C) If the Retrieval FU is supported:

When *ma\_start\_retrieval()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

### 5.3.9 Submit()

#### 5.3.9.1 Synopsis

R13 If the Submission FU is supported:

When the header file <xmh.h> is included, then the function *ma\_submit()* is declared as follows:

```
OM_return_code
ma_submit (
    OM_private_object      session,
    OM_object              communique,
    OM_private_object      *submission_results
);
```

#### 5.3.9.2 Results

014(D) If the Submission FU is supported:

When *ma\_submit()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

### 5.3.10 Wait()

#### 5.3.10.1 Synopsis

R14 If the Delivery or Retrieval FU is supported:

When the header file `<xmh.h>` is included, then the function *ma\_wait()* is declared as follows:

```
OM_return_code
ma_wait (
    OM_private_object      session,
    MH_interva             interval,
    MH_sequence_number     minimum_sequence_number,
    OM_boolean             *available
);
```

#### 5.3.10.2 Results

015(D) If the Delivery FU is supported:

When *ma\_wait*(*,minimum\_sequence\_number,*) is called and completes successfully, then *minimum\_sequence\_number* is ignored by the service.

#### 5.3.10.3 Results

016(D) If the Delivery or Retrieval FU is supported:

When *ma\_wait()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

## 5.4 Transfer Functions

### 5.4.1 Close

#### 5.4.1.1 Synopsis

R15 If the Basic Transfer FU is supported:

When the header file `<xmh.h>` is included, then the function *mt\_close()* is declared as follows:

```
OM_return_code
mt_close (
    OM_private_object      session
);
```

#### 5.4.1.2 Results

017(D) If the Basic Transfer FU is supported:

When *mt\_close()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

## 5.4.2 Finish Transfer In

### 5.4.2.1 Synopsis

- R16 If the Transfer In FU is supported:  
When the header file `<xmh.h>` is included, then the function `mt_finish_transfer_in()` is declared as follows:

```
OM_return_code
mt_finish_transfer_in (
    OM_private_object    session,
    OM_private_object    object,
    OM_boolean           remove
);
```

### 5.4.2.2 Description

- 018(C) If the Transfer In FU is supported:  
When `mt_finish_transfer_in()` is called, then the *all* argument is true if and only if the object argument is the null pointer.

### 5.4.2.3 Results

- 019(D) If the Transfer In FU is supported:  
When `mt_finish_transfer_in()` is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

## 5.4.3 Open

### 5.4.3.1 Synopsis

- R17 If the Basic Transfer FU is supported:  
When the header file `<xmh.h>` is included, then the function `mt_open()` is declared as follows:

```
OM_return_code
mt_open (
    OM_string            client_name,
    OM_string            client_instance_name,
    MH_feature           feature_list[],
    OM_private_object    *session,
    OM_workspace         *workspace
);
```

### 5.4.3.2 Description

- 020(C) If the Basic Access FU is supported:  
When `mt_open(,feature_list,,)` is called and completes successfully, then the *feature\_list* sequence is terminated by an object identifier having a length of zero and any value for the data pointer.

### 5.4.3.3 Results

- 021(D) If the Basic Transfer FU is supported:  
When `mt_open()` is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

## 5.4.4 Size

### 5.4.4.1 Synopsis

R18 If the Transfer In FU is supported:

When the header file <xmh.h> is included, then the function *mt\_size()* is declared as follows:

```
OM_return_code
mt_size (
    OM_private_object    session,
    OH_object_count      *number
);
```

### 5.4.4.2 Results

022(D) If the Transfer In FU is supported:

When *mt\_size()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

## 5.4.5 Start Transfer In

### 5.4.5.1 Synopsis

R19 If the Transfer In FU is supported:

When the header file <xmh.h> is included, then the function *mt\_start\_transfer\_in()* is declared as follows:

```
OM_return_code
mt_start_transfer_in (
    OM_private_object    session,
    OM_private_object    *object,
    OM_string             *identifier
);
```

### 5.4.5.2 Results

023(D) If the Transfer In FU is supported:

When *mt\_start\_transfer\_in()* is called and an invalid pointer was supplied as a function argument or as the receptacle for a function result and no other error condition applies, then it will return either [OM\_SUCCESS] or the [OM\_POINTER\_INVALID] return code. (See GA01 in 5.5.)

## 5.4.6 Transfer Out

### 5.4.6.1 Synopsis

R20 If the Transfer Out FU is supported:

When the header file <xmh.h> is included, then the function *mt\_transfer\_out()* is declared as follows:

```
OM_return_code
mt_transfer_out (
    OM_private_object    session,
    OM_object             object
);
```