X/Open CAE Specification

Message Store API (XMS)

X/Open Company Ltd.

X/Open CAE Specification

Message Store API (XMS)

ISBN: 1 872630 83 9

X/Open Document Number: C305

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

List of Figures

List of Tables

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and allows users to move between systems with a minimum of retraining.

The components of the Common Applications Environment are defined in X/Open CAE Specifications. These contain, among other things, an evolving portfolio of practical application programming interfaces (APIs), which significantly enhance portability of application programs at the source code level, and definitions of, and references to, protocols and protocol profiles, which significantly enhance the interoperability of applications.

The X/Open CAE Specifications are supported by an extensive set of conformance tests and a distinct X/Open trademark - the XPG brand - that is licensed by X/Open and may be carried only on products that comply with the X/Open CAE Specifications.

The XPG brand, when associated with a vendor's product, communicates clearly and unambiguously to a procurer that the software bearing the brand correctly implements the corresponding X/Open CAE Specifications. Users specifying XPG-conformance in their procurements are therefore certain that the branded products they buy conform to the CAE Specifications.

X/Open is primarily concerned with the selection and adoption of standards. The policy is to use formal approved *de jure* standards, where they exist, and to adopt widely supported *de facto* standards in other cases.

Where formal standards do not exist, it is X/Open policy to work closely with standards development organisations to assist in the creation of formal standards covering the needed functions, and to make its own work freely available to such organisations. Additionally, X/Open has a commitment to align its definitions with formal approved standards.

X/Open Specifications

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) Specifications are the long-life specifications that form the basis for conformant and branded X/Open systems. They are intended to be used widely within the industry for product development and procurement purposes.

Developers who base their products on a current CAE Specification can be sure that either the current specification or an upwards-compatible version of it will be referenced by a future XPG brand (if not referenced already), and that a variety of compatible, XPG-branded systems capable of hosting their products will be available, either immediately or in the near future.

CAE Specifications are not published to coincide with the launch of a particular XPG brand, but are published as soon as they are developed. By providing access to its specifications in this way, X/Open makes it possible for products that conform to the CAE (and hence are eligible for a future XPG brand) to be developed as soon as practicable, enhancing the value of the XPG brand as a procurement aid to users.

- *Preliminary Specifications*

  These are specifications, usually addressing an emerging area of technology, and consequently not yet supported by a base of conformant product implementations, that are released in a controlled manner for the purpose of validation through practical implementation or prototyping. A Preliminary Specification is not a ''draft'' specification. Indeed, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE Specification.

  Preliminary Specifications are analogous with the ''trial-use'' standards issued by formal standards organisations, and product development teams are intended to develop products on the basis of them. However, because of the nature of the technology that a Preliminary Specification is addressing, it is untried in practice and may therefore change before being published as a CAE Specification. In such a case the CAE Specification will be made as upwards-compatible as possible with the corresponding Preliminary Specification, but complete upwards-compatibility in all cases is not guaranteed.

In addition, X/Open periodically publishes:

- *Snapshots*

  Snapshots are ''draft'' documents, which provide a mechanism for X/Open to disseminate information on its current direction and thinking to an interested audience, in advance of formal publication, with a view to soliciting feedback and comment.

  A Snapshot represents the interim results of an X/Open technical activity. Although at the time of publication X/Open intends to progress the activity towards publication of an X/Open Preliminary or CAE Specification, X/Open is a consensus organisation, and makes no commitment regarding publication.

  Similarly, a Snapshot does not represent any commitment by any X/Open member to make any specific products available.

X/Open Guides

X/Open Guides provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant.

X/Open Guides are not normative, and should not be referenced for purposes of specifying or claiming X/Open-conformance.

This Document

This document is a CAE Specification (see above).

The Message Store Application Program Interface provides an API to the Message Store functions similar to those described in X.413 in an X.400 Message Handling System. This document describes this program interface.

A compliant system shall meet the definitive requirements described in this Message Store API Preliminary Specification.

The content of this specification has been developed in collaboration with the X.400 API Association. This is one of several specifications that X/Open has developed in collaboration with the X.400 API Association. Other documents include:

- OSI-Abstract-Data Manipulation (XOM) API
- API to Directory Services (XDS)
- API to Electronic Mail (X.400)
- Guide to Selected X.400 and Directory Services APIs
- EDI Messaging Package.

Trade Marks

X/Open and the 'X' device are trademarks of X/Open Company Limited in the U.K. and other countries.

Referenced Documents

The following documents are referenced in this Specification:

ANSI C

Information Processing - Programming Language C, ISO Draft International Standard DIS 9899 (also known as ‘‘ANSI C’’, American National Standard X3.159 -1989).

MHS-1984

CCITT X.400 (1984) includes the following:

Recommendation X.400, Message Handling Systems: System Model - Service Elements, International Telegraph and Telephone Consultative Committee (CCITT) Red Book, Fascicle VIII.7, International Telecommunications Union, 1984, pp. 3-38.

Recommendation X.401, Message Handling Systems: Basic Service Elements and Optional User Facilities, Ibid., pp. 39-45.

Recommendation X.408, Message Handling Systems: Encoded Information Type Conversion Rules, Ibid., pp. 46-61.

Recommendation X.409, Message Handling Systems: Presentation Transfer Syntax and Notation, Ibid., pp. 62-93.

Recommendation X.410, Message Handling Systems: Remote Operations and Reliable Transfer Service, Ibid., pp. 93-126.

Recommendation X.411, Message Handling Systems: Message Transfer Layer, Ibid., pp. 127-182.

Recommendation X.420, Message Handling Systems: Interpersonal Messaging User Agent Layer, Ibid. pp 182-219.

Recommendation X.430, Message Handling Systems: Access Protocol for Teletex terminals, Ibid. pp 219-266.

MHS-1988

CCITT X.400 (1988) includes the following:

Recommendation X.400, Message Handling Systems: System Model - Service Elements, International Telegraph and Telephone Consultative Committee (CCITT) Blue Book, Fascicle VIII.7, International Telecommunications Union, 1988. See also ISO 10021-1.

Recommendation X.402, Message Handling Systems: Overall Architecture, Ibid. See also ISO 10021-2.

Recommendation X.403, Message Handling Systems: Conformance Testing, Ibid.

Recommendation X.407, Message Handling Systems: Abstract Service Definition Conventions, Ibid. See also ISO 10021-3.

Recommendation X.408, Message Handling Systems: Encoded Information Type Conversion Rules, Ibid.

Recommendation X.411, Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures, Ibid. See also ISO 10021-4.

Recommendation X.413, Message Handling Systems: Message Store: Abstract Service Definition, Ibid. See also ISO 10021-5.

Recommendation X.419, Message Handling Systems: Protocol Specifications, Ibid. See also ISO 10021-6.

Recommendation X.420, Message Handling Systems: Interpersonal Messaging System, Ibid. See also ISO 10021-7.

MHS

MHS Implementor's Guide, Version 5, CCITT Special Rapporteur Q18/VII Message Handling Systems, February 1991.

XDS

API to Directory Services (XDS), CAE Specification, X/Open Company Limited and X.400 API Association, C190, 1991.

XOM

OSI-Abstract-Data Manipulation (XOM) API, CAE Specification, X/Open Company Limited and X.400 API Association, C180, 1991.

XSH

X/Open Portability Guide, Issue 4 (XPG4), System Interfaces and Headers, CAE Specification, C203, X/Open Company Limited, 1992.

X.400

API to Electronic Mail (X.400), CAE Specification, X/Open Company Limited and X.400 API Association, C191, 1991.

X.411

Recommendation X.411, Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures, International Telegraph and Telephone Consultative Committee (CCITT) Blue Book, Fascicle VIII.7, International Telecommunications Union, 1988. See also ISO 10021-4.

X.413

Recommendation X.413, Message Handling Systems: Message Store: Abstract Service Definition and Procedures, Ibid. See also ISO 10021-5.

X.420

Recommendation X.420, Message Handling Systems: Message Store: Interpersonal Messaging System, Ibid. See also ISO 10021-7.

Chapter 1

Introduction

1.1 Overview

The Message Store Application Program Interface (MS API) provides an Application Program Interface (API) to the Message Store (MS) functions similar to those described in X.413 (see reference **X.413**) in an X.400 Message Handling System. A brief description of the MS is given in Section 1.3 on page 3.

This interface is designed to offer services that are consistent with, but not limited to, the 1988 CCITT X.413 Recommendations and the ISO 10021-5 Standard. The CCITT Recommendations and the ISO Standard were developed in close collaboration and are technically aligned. Hereafter, they are referred to as *the Standards*.

The interface is designed for operational interactions with a Message Store. Although the semantics of the interface are derived from X.413, this specification does not require for conformance that an implementation of the interface or the MS itself actually make use of the MS (P7) protocol of X.413.

The MS interface uses facilities provided by the XOM Specification (see reference **XOM**). Section 1.6 on page 9 presents some of the important definitions used in the XOM API.

1.2 Abbreviations

The objects defined in this document are to be understood in the context provided by the MS standards X.413 (see reference **X.413**).

In addition to the abbreviations used to identify the Referenced Documents, the following are abbreviations used throughout this document:

API	Application Program Interface
ASN.1	Abstract Syntax Notation One
AVA	Attribute Value Assertion
BER	Basic Encoding Rules
CCITT	International Telegraph and Telephone Consultative Committee
FU	Functional Unit
IA5	International Alphabet Number 5
ID	identifier
IM	Interpersonal Messaging
IPM	interpersonal message
IPN	interpersonal notification
ISO	International Organization for Standardization
MS	Message Store
OM	Object Management
OSI	Open Systems Interconnection
X.400 APIA	X.400 API Association

1.3 The Message Store

The *Message Store* (MS) Abstract Service is defined in X.413 (see reference **X.413**). The MS acts as an intermediary between the *Message Transfer System* (MTS) and the *User Agent* (UA). The main function of the MS is to accept delivery of messages on behalf of a single end-user and to store the messages for subsequent retrieval by the end-user's UA. The MS also makes available the message submission and administration services of the MTS to the UA. In this intermediate capacity, the MS can provide the UA with additional functionality compared to direct submission to the MTS. The indirect submission services offered by the MS also includes forwarding of messages stored in the MS.

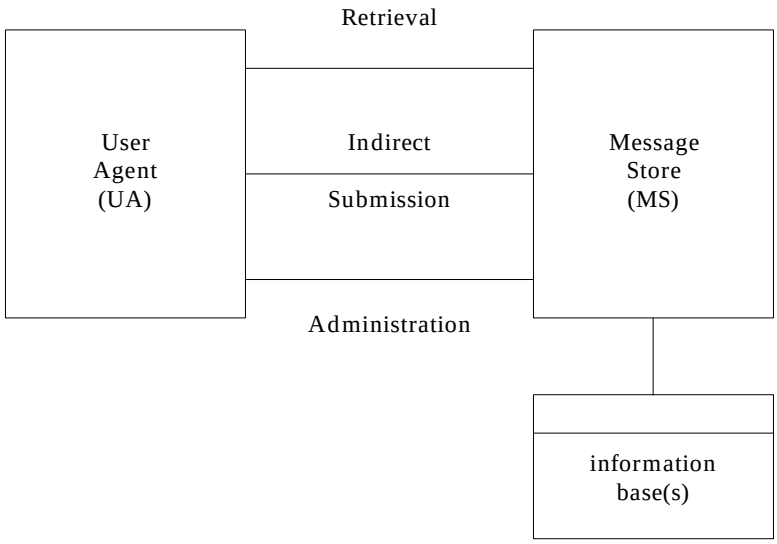

Figure 1-1 MS and UA Interface Operations

The MS Abstract Service defines operations that can be grouped into two sets.

One set interfaces to the MTS and provides the Delivery, Submission and Administration services. This set corresponds to the X.400 *P3 protocol*.

The other set of operations interfaces with the UA and corresponds to the *P7 protocol* (refer to Figure 1-1). These operations, in turn, may be classified thus:

- **Retrieval**
  - — list
  - — fetch
  - — delete
  - — register-MS
  - — alert
  - — summarise

- **Indirect Submission**
  - — submit message
  - — submit probe
  - — cancel submission
- **Administration**
  - — register
  - — change credentials
    Note: This API does not include a function for Change-Credentials.

In addition, the following are provided:

- MS-Bind
- MS-Unbind

The P7 protocol gives a UA access to the MS. The MS allows the UA to submit messages, to retrieve messages (or parts of such messages) and to perform administrative functions such as registration of the MS (and/or UA) capabilities.

The MS stores and maintains databases known as *information bases*. The Standards define three types of information bases: *stored messages* (for delivered messages and reports), *inlog* and *outlog* (these last two are for logging purposes).

An MS may provide special support for various types of messages, such as interpersonal messages (IPMs). X.413 discusses how to support IPMs.

The elements in an information base of the MS are known as *entries*. An *entry* represents a single object (such as a delivered message) within the information base. An entry consists of a set of *MS attributes*. Each entry is identified by the MS attribute *Sequence-Number* which is unique within an information base and generated when a new entry is created.

An *MS attribute* provides a piece of information about, or derived from, the data to which the entry corresponds. An attribute consists of an *attribute type* which identifies the class of information given by an attribute and its corresponding *attribute value*(s) which are particular instances of that class appearing in the entry. Attributes that can have exactly a single value associated with them are termed *single-valued*, whereas those that can have more than one value at a time are termed *multi-valued* (e.g., Other-Recipient-Names). The standards define certain general-purpose attributes, termed *General Attributes*, for the stored messages information bases (see reference **X.413**, Clause 11). Examples of such MS General Attributes types are Entry-Type, Sequence-Number, Originator-Name and Report-Delivery-Envelope. An MS that supports Interpersonal Messaging (IM) will also recognise specific IM attributes and can take appropriate actions on these attributes.

1.4 Mandatory and Optional Features of the Interface

This document defines an API that application programs can use to access the functionality of the underlying MS service. The interface neither defines nor implies any particular profile of the service.

Note that nothing in this specification requires that the implementation of the interface or the Message Store itself actually make use of the P7 protocol.

Options

Some behavioural aspects of the interface are implementation-defined. These are:

- the maximum number of outstanding asynchronous operations
- whether an asynchronous function call returns before the operation is submitted to the MS
- the text and language of error messages

The *features* that the present edition of this document defines are the functional units and *packages* as shown in Table 1-1 below.

A *functional unit* (FU) is a collection of related functions. If an FU is supported by the service, all interface functions within that FU must be supported.

See Section 1.6.6 on page 11 for the definition of a package.

Other documents, or other editions of this document, may define additional (e.g., proprietary) features which may (but need not) themselves be FUs or packages.

Feature	**Object Identifier (ASN.1) suffix †**	**mandatory/ optional**
MS Package	ms-pkg(1)	mandatory
MS General Attributes Package	ms-general-attr-pkg(2)	mandatory
MS IM Attributes Package	ms-im-attr-pkg(3)	optional
MS FU	ms-fu(4)	mandatory
MS Submission FU	ms-submission-fu(5)	optional
MS Administration FU	ms-administration-fu(6)	optional
MS Alert FU	ms-alert-fu(7)	optional

† Each object identifier suffix is appended to:
{joint-iso-ccitt mhs-motis(6) group(6) white(1) api(2) ms(6)}

Table 1-1 Features and their Object Identifiers

This specification defines four FUs. See Section 4.2 on page 35 for the table listing the interface functions available within each of these functional units.

This specification defines three packages, of which two are mandatory and one is optional. Use of optional packages is negotiated through the *Feature-List* argument of the **Initialize**( ) function. The packages defined are as follows:

- The *Message Store Package*, defined in Chapter 5, with the *Errors* defined in Chapter 6, is mandatory.
- The *Message Store General Attributes Package*, defined in Chapter 7, is mandatory.
- The *Message Store Interpersonal Messaging Attributes Package*, defined in Chapter 8, is optional.

The service makes an optional feature available to the client only if the client requests it. The client may request a set of features through the *Feature-List* argument to the **Initialize**( ) function; the MS FU, the MS Package and the MS General Attributes Package are provided by the service

as default.

The following occurs if the client attempts to use a feature that the service has not made available. If the MS FU is available and the client invokes a function in the other FUs, but that FU is unavailable, the

feature-unavailable [MS_E_FEATURE_UNAVAILABLE]

error arises.

After initialising the interface with certain features negotiated, if the client attempts to invoke a function not previously requested, the

feature-not-requested [MS_E_FEATURE_NOT_REQUESTED]

error arises.

If the client supplies, as a function argument, an object, one of whose subobjects is an instance of a class not in any of the packages the interface had been initialised with, the

no-such-class [MS_E_NO_SUCH_CLASS]

error arises.

This specification does not mandate that any OM classes be encodable using **OM-Encode**( ) and **OM-Decode**( ).

1.5 Conformance

A manufacturer shall claim conformance to this edition of this document only if it and its product collectively satisfy the following requirements:

1. Interfaces

   The manufacturer and product shall satisfy the following requirements related to interfaces:

   — The manufacturer shall identify the interface the product implements and state what role the product plays for the client and/or the service.

   — The product shall implement the OM interface as defined in the XOM API Specification (see reference **XOM**), satisfying its conformance requirements and play the same roles for that interface as it plays for the MS interface.

2. Features

   The manufacturer and product shall satisfy the following requirements related to features (see Section 1.4 on page 5):

   — If the product plays the role of service for the MS interface, the manufacturer shall state which features it implements.

   — If the product plays the role of service for the MS interface, it shall implement the MS FU and optionally, the Alert FU, the Submission FU, the Administration FU or any combination of the three.

   — If the product plays the role of service for the MS interface, it shall implement the MS Package, MS General Attributes Package and optionally, the MS Interpersonal Messaging Package.

3. Functions

   The manufacturer and product shall satisfy the following requirements related to functions:

   — The product shall implement every aspect of every function in each FU for which it plays the role of service.

4. Classes

   The manufacturer and product shall satisfy the following requirements related to classes:

   — Support for any of the packages defined in this specification only requires the service to provide those classes which can be supported by the underlying implementation of the MS.

   Note: For example, not all implementations of an MS support all the MS attributes defined in X.413 (see reference **X.413**). Therefore, the service need not support the OM classes corresponding to these unsupported MS attributes.

   — The product shall implement the closures of all classes it implements.

   — The product shall state classes for which it provides the **OM-Encode**( ) and **OM-Decode**( ) functions.

5. Protocols

   The manufacturer and product shall satisfy the following requirements related to protocols:

— If the product plays the role of service for the MS interface, the manufacturer shall state whether or not it realises the interface by means of the X.413 Message Store (P7) Protocol.

— If the product implements the P7 protocol, the manufacturer and product shall satisfy the conformance requirements of the X.400 (1988) and relevant profiles with respect to the protocol.

6. Options

   The manufacturer and product shall satisfy the following requirements related to implementation (see Section 1.4 on page 5):

   — If the product plays the role of service for the MS interface, the manufacturer shall state the behaviour of implementation-defined options.

7. Interpretation of ‘‘any’’ syntax

   Wherever the word ‘‘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 the XOM API Specification (see reference **XOM**), otherwise it shall be treated as String(Encoding).

1.6 Object Management

The interface makes use of facilities provided by the XOM API (see reference **XOM**). These facilities are introduced briefly below.

Note that some terms used (e.g., attribute) are also used in a different context when referring to the Message Store. To avoid confusion, distinct names are used for each such term. Throughout this document, care is taken to distinguish between OM attributes (refer to Section 1.6.3) and attributes used with regard to the Message Store (refer to Section 1.3 on page 3). The unqualified term *attribute* denotes the Message Store construct, whereas the phrase *OM attribute* denotes the Object Management one.

1.6.1 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 Object Management specification and nowhere else, and are themselves represented by integers.

1.6.2 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).

1.6.3 OM Attribute

An *OM attribute type* is an arbitrary category into which a specification places some values.

OM attribute types are represented by integers assigned in the individual API service specifications and are only meaningful within a particular package (see Section 1.6.6 on page 11).

An *OM attribute* is an OM attribute type together with an ordered sequence of 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 Section 1.6.5 on page 10).

The OM attribute type can be thought of as the name of the OM attribute.

There is no general representation for an OM attribute (see Section 1.6.9 on page 11), but a descriptor represents an OM attribute type together with a single syntax and value.

1.6.4 OM Object

An *OM object* is a collection of OM attributes, the values of which can be accessed by means of functions. The particular OM attribute types that may occur in an OM object are determined by the OM class (see Section 1.6.5 on page 10) of the OM object 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. The internal representation of an OM object is not specified though there is a defined data structure called a *descriptor list* which can also be used directly in a program (see Section 1.6.9 on page 11).

1.6.5 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. 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.

Due to 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 be 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 specification of a class may impose arbitrary constraints on its attributes. For instance, the more common of these include the constraints to:

- restrict the syntaxes permitted for values of an attribute (often to a single syntax)
- restrict the particular values to a subset of those permitted by the syntax
- require one or more values of the attribute (i.e., a mandatory attribute)
- allow either zero or more values of the attribute (i.e., an optional attribute)
- permit multiple values, perhaps up to some limit known as the value number constraint
- restrict the length of strings (in octets), up to a limit known as the *value length constraint*.

Constraints may affect multiple attributes at a time; e.g., a rule that only one of a set of several 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.

In contrast to abstract classes, a *concrete class* is an OM class of which instances are permitted.

1.6.6 Package

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

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.

1.6.7 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 another package. This is done to share definitions and to avoid duplication.

A *Package-Closure* is a set of OM classes which 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 the XOM API Specification - see reference **XOM**).

1.6.8 Workspace

Details of the representation of OM objects and of the implementation of the functions 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.

A *workspace* is one or more Package-Closures, together with an implementation of the Object Management functions that supports 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 the XOM API Specification (see reference **XOM**) for further details on how workspaces are implemented.

1.6.9 Descriptor

A *descriptor* is a defined data structure that is used to represent an OM attribute type and a single value. The structure has three components: a type, a syntax and a value.

A *descriptor list* is an ordered sequence of descriptors that is used to represent several OM attribute types and values.

Where the list contains several descriptors with the same OM attribute type (representing a multi-valued OM attribute), the order of the values in the OM attribute is the same as the order in the list. Such descriptors will always be adjacent.

Where the list contains a descriptor representing the OM class, this must occur before any others.

A *public object* is a descriptor list that contains all the OM attribute values of an OM object, including the OM class. Public objects are used to simplify client programs by enabling the use of static data structures instead of a sequence of OM function calls.

A *private object* is an OM object created in a workspace using the OM functions or the functions provided by an application-specific API. The term is simply used for contrast with a public object.

1.6.10 Use of Objects

OM objects are used to represent data collections used in the interface, such as a message or function results.

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 client can supply a subclass and that the service can return a subclass. For example, the client can submit messages in any format which is defined as a subclass of the class *Submitted-Communique* and the service can return an error in any of the subclasses of the abstract class, *Error*.

Since the service may return a subclass of the specified OM class, the client should always use the **OM-Instance**( ) function when checking the OM class of an OM object, rather than testing the value of the Class OM attribute.

The subclassing mechanism is used within this specification to allow different specialisations of a class to be used in the same manner in an interface. Additional specifications may define packages containing subclasses which further specialise interface classes. These packages may be for specific application domains (e.g., EDI Messaging classes) or for specific vendor products. When the client supplies a subclass of a specified OM class as an argument, the service either will recognise the subclass as an OM class of a service-supported package or will ignore all OM attribute types which are not permitted in that OM class.

The client can generally supply either a public object or private object as an argument of the interface functions. 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 client 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 must not be modified in any way.

1.7 Conventions Used in this Specification

This specification describes a programming language-independent interface to the Message Store together with a specific C language binding of this interface.

Certain conventions are used to identify particular items pertaining to this interface:

- Items in **bold font** are language-independent names and are spelled with hyphens between words. The first letter of function names and arguments, OM class names and OM attributes are capitalised (e.g., **Completion-Flag**), whereas the names of constants are in lower-case (e.g., **completed-operation**). The names of functions are followed by parentheses (e.g., **Bind**( )).
- Items in *italic font* spelled with underscores between words are either C language names or the names of abstract OM classes. The names of errors are enclosed in brackets (e.g., *[MS_E_NOSYS]*), whereas the names of other constants are enclosed in braces (e.g., *{MS_COMPLETED_OPERATION}*). The names of functions are followed by parentheses (e.g., *ms_bind*( )). More details of the C language binding are given in Chapter 2. *Italics* are also used for emphasis and in particular when introducing key terms.

Chapter 2

C Language Binding

2.1 Introduction

This chapter is relevant for client programs written in the C language since it describes certain characteristics of the C language binding to the Message Store interface. This chapter covers function names, type definition (i.e., ''typedef'') names and constants. All the C identifiers are derived from the language-independent names as explained below. There is a complete list of all the identifiers in Chapter 9. For ease of use, some of these identifiers are defined in the specification along with the language-independent names.

All C language names are set in an italic typeface; also a **function**( ) is indicated by its trailing parentheses while a *{CONSTANT}* is sandwiched by braces except for names of *[ERRORS]* where each is sandwiched by brackets.

The definitions of the C identifiers appear in these header files:

<xom.h> with definitions for the associated OM interface (see reference **XOM**)

<xomi.h> with definitions for the workspace interface (see reference **XOM**)

<xms.h> with definitions for the Message Store interface

<xmsga.h> with definitions for the Message Store General Attributes

<xmsima.h> with definitions for the Message Store Interpersonal Messaging Attributes

2.2 C Naming Convention

The interface uses part of the C public namespace for its facilities. All identifiers start with the letters *ms*, *MS* or *OMP*; more details of the conventions used are given in the following table Table 2-0). Note that the interface reserves all identifiers starting with the letters *msP* for private (i.e., internal) usage by implementations of the interface. It also reserves all identifiers starting with the letters *msX* or *MSX* for vendor-specific extensions of the interface. Hence, client programs should not use any identifier starting with these reserved letters.

The Object Management API uses similar, though not identical, naming conventions described in the XOM Specification (see reference **XOM**). All its identifiers are prefixed by the letters *om* or *OM*.

Item	Prefix
reserved for implementors	*msP*
reserved for implementors	*OMP*
reserved for interface extensions	*msX*
reserved for interface extensions	*MSX*
<xms.h>	
functions	*ms_*
error 'problem' values	*MS_E_*
OM class names	*MS_C_*
OM value length limits	*MS_VL_*
OM value number limits	*MS_VN_*
other constants	*MS_*
<xmsga.h>	
MS attribute types	*MS_A_*
<xmsima.h>	
MS IM attribute types	*MS_IM_A_*

Table 2-1 C Naming Conventions

A complete list of all identifiers used (except those beginning with *msP*, *msX*, *MSX* or *OMP*) are given in Chapter 9. No implementation of the interface will use any other public identifiers.

The C identifiers are derived from the language-independent names used throughout this specification by a systematic process which depends on the kind of name:

- Function names are entirely composed of lower-case letters and prefixed by *ms_*. Thus, **Receive-Result**( ) becomes *ms_receive_result*( ).
- C function parameters are derived from the argument and result names by making them entirely lower-case. In addition, the names of results are suffixed by *_return*. Thus, the argument **Selector** becomes selector while the result **Invoke-ID** becomes *invoke_id_return*.
- OM class names are entirely composed of upper-case letters and prefixed by *MS_C_*. Thus, **Fetch-Result** becomes *MS_C_FETCH_RESULT*.
- Enumeration tags are derived from the name of the corresponding OM syntax by prefixing with *MS_*. The case of the letters is left unaltered. Thus, **Enum(Problem)** becomes *MS_Problem*.
- Enumeration constants, as well as the names of OM attributes and all other constants except errors are entirely composed of upper-case letters and prefixed by *MS_*. Thus, **Stored-**

Messages becomes *MS_STORED_MESSAGES*.

- Errors are treated as a special case. Constants that are the possible values of the OM attribute **Problem** of a subclass of the OM class **Error** are entirely composed of upper-case letters and are prefixed by *MS_E_*. Thus, **delete-restriction-problem** becomes *MS_E_DELETE_RESTRICTION_PROBLEM*.
- Where names exceed 31 characters in length, the C binding names are abbreviated to be unique within the first 31 characters without affecting the alphabetical ordering of names with the same *MS_* prefix.
- The constants in the *Value Length* and *Value Number* columns of the OM class definition tables are also assigned identifiers. (They have no names in the language-independent specification.) Where the upper limit in one of these columns is not '1' (one), it is given a name consisting of the OM attribute name prefixed by *MS_VL_* for value length or *MS_VN_* for value numbers.
- The sequence of octets for each object identifier is also assigned an identifier for internal use by certain OM macros. These identifiers are all upper-case letters and are prefixed by *OMP_O_*. The XOM Specification (see reference **XOM**) gives further details on the use of object identifiers.

Note that hyphens are translated everywhere to underscores.

2.3 Use and Implementation of Interfaces

Chapter 3 contains the detailed descriptions for the interface functions. The following statements in this subsection apply unless explicitly stated otherwise in Chapter 3.

If an argument to a function has an invalid value (such as a value outside the domain of the function or a pointer outside the address space of the program), the behaviour is undefined.

Any function declared in a header may be implemented as a macro defined in the header so a library function should not be declared explicitly if its header is included.

Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses because the name is not then followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a library function even if it is also defined as a macro. The use of *#undef* to remove any macro definition will also ensure that an actual function is referred to. Any invocation of a library function that is implemented as a macro will expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary; hence it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression wherever a function with a compatible return data type could be called.

2.4 Function Return Values

The return value of a C function is always bound to the **Status** result of the language-independent description. Functions return a value of type *MS_status*, which is an error indication. If, and only if, the function succeeds, its value will be **success**, expressed in C by the constant *{MS_SUCCESS}*. If a function returns with a status other than this, then it has not updated the return parameters. The value of the status, in this case, is an error as described in Chapter 6.

Since C does not provide multiple return values, functions must return all other results by writing into storage passed by the client program. Any argument that is a pointer to such storage has a name ending with *_return*. For example, the C parameter declaration *Uint *completion_flag_return* indicates that the function will return an unsigned integer **Completion-Flag** as a result, so the actual argument to the function must be the address of a suitable variable. This notation allows the reader to distinguish between an input parameter that happens to be a pointer and an output parameter where the '*' is used to simulate the semantics of passing by reference.

2.5 Compilation and Linking

All client programs that use this interface must include **<xom.h>**, **<xms.h>** and **<xmsga.h>** headers in this order. If the optional MS Interpersonal Messaging Attributes Package is supported, the **<xmsima.h>** header can be included after the other headers.

See Appendix A for details on run-time binding and the naming conventions for the libraries.

Chapter 3

Description

3.1 Introduction

The interface comprises a number of functions together with many OM classes of OM objects which are used as the arguments and results of the functions. Both the functions and the OM objects are based closely on the Message Store Abstract Service as specified in X.413 (see reference **X.413**).

The interface models interactions with the Message Store as service requests made through a number of interface *functions* which take a number of input *arguments*. Each valid request causes an *operation* within the Message Store which eventually returns a *status* and any *result* from the operation.

All interactions between the user and the MS belong to a *session* which is represented by an OM object passed as the first argument to most interface functions.

The MS Package (see Chapter 5 and Chapter 6) defines the classes that describe the arguments to the interface functions. The Indirect Submission and Administration operations also require the class definitions of the Message Handling Package (see reference **MHS**) to be imported.

The MS General Attributes Package (see Chapter 7 defines the classes that describe the MS General Attributes. This package is needed to access the General Attributes of entries in the MS.

The MS Interpersonal Messaging Attributes Package (see Chapter 8) defines the classes that describe the MS IM Attributes. If the client requires to view and access the MS entries with individual IM attributes, this package must be supported. Even if the MS IM Attributes Package is not supported, it is possible to use the interface to retrieve the message content as a whole (encoded in ASN.1) but not the individual IM attributes.

The main features of the interface are described in the rest of this chapter.

3.2 Services

The Standards define the Abstract Service that the User Agent uses to interact with the Message Store. Each operation of this Abstract Service maps to a single interface function with the same name. Detailed specifications for these interface functions are given in Chapter 4.

In addition, there is a function, **Receive-Result**( ), which has no counterpart in the Message Store Abstract Service. This function is used in conjunction with asynchronous operations and is explained in Section 3.6 on page 28.

The interface functions, **Check-Alert**( ), **Wait**( ), **Initialize**( ) and **Shutdown**( ) also have no counterparts in the Message Store Abstract Service.

The interface functions are summarised in the table below Table 3-0). Functions that can execute asynchronously are indicated by an 'a' in the table; all other functions always execute synchronously.

	Name	Description
	Bind	Establish a session with the Message Store.
a	**Cancel-Submission**	Cancel a message submitted with the deferred delivery option.
	Check-Alert	Check if the MS has received new entries whose attributes match the criteria previously supplied by the **Register-MS**( ) function.
a	**Delete**	Remove selected entries from an information base.
a	**Fetch**	Get information on a specific entry in an information base.
	Initialize	Initialise the interface, returning a workspace.
a	**List**	Return selected information for a list of entries of interest from an information base.
	Receive-Result	Retrieve the result of an asynchronously executed operation.
a	**Register**	Modify various parameters held by the MTS regarding delivery of messages to the MS.
a	**Register-MS**	Register or deregister various information with the MS.
	Shutdown	Shut down the interface, discarding the workspace.
a	**Submit**	Submit a communique (message or probe).
a	**Summarize**	Summarise counts of selected entries in an information base.
	Unbind	Terminate a session with the Message Store.
a	**Wait**	Return when a new entry is available in the Message Store for retrieval or when a specified period of time has elapsed, whichever occurs first.

Table 3-1 Interface Functions

3.2.1 Sequence of Interface Functions

The interface has an initialisation and shutdown sequence that allows the negotiation of optional features. This involves the functions **Initialize**( ) and **Shutdown**( ).

Every client program must first call **Initialize**( ) which returns a workspace. By default, this workspace supports the standard MS Package (see Chapter 5 and Chapter 6) and the MS General Attributes Package (see Chapter 7). The workspace can be extended to support the optional MS Interpersonal Messaging Attributes Package (see Chapter 8) or any vendor extensions. Vendor extensions may include additional packages and may also include additional or modified functionality. All such packages or other extensions are identified by means of OSI Object Identifiers.

Once the interface has been initialised, **Initialize**( ) may not be called again until **Shutdown**( ) has been called. Figure 3-0 depicts the order and context in which each interface function can be used.

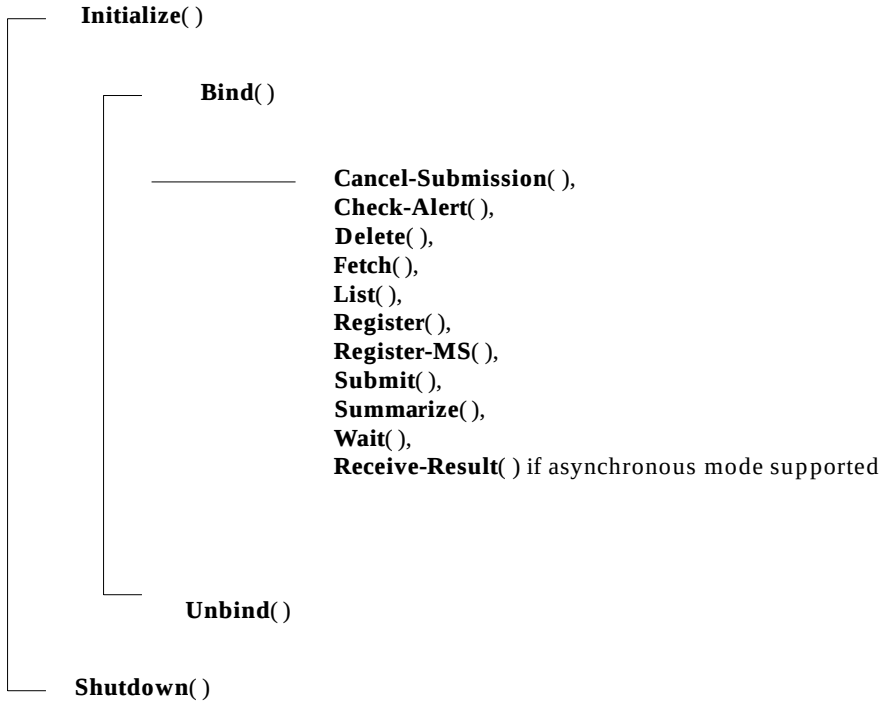

Figure 3-1 Sequence of Interface Functions

After negotiating a workspace with the required features, the client can use the workspace as required. It can create and manipulate OM objects using the OM functions and can start one or more MS sessions using **Bind**( ).

Each session is terminated using **Unbind**( ), and all its OM objects released using **OM-Delete**( ); finally the client should ensure that resources associated with the interface are released by calling **Shutdown**( ).

It is possible to retain access to service-generated public objects after **Shutdown**( ) has been called or to start another cycle by calling **Initialize**( ) if desired.

3.3 Session

A session provides information about a particular association between a client program and the service. A session identifies the association (or connection) with the MS over which an MS operation is to be sent. A session is commenced by a successful **Bind**( ) function and terminated by the **Unbind**( ) function. Multiple sessions are allowed, if the maximum number of sessions negotiated through **Initialize**( ) is more than one.

A session is described by an OM object of the class **Session**. An instance of this class is returned by the **Bind**( ) function and this is subsequently passed as the first argument to most interface functions.

Detailed specifications of the OM class **Session** are given in Chapter 5.

3.4 Function Arguments

Each interface function takes a number of input arguments (also called input parameters). **Session** is an argument to several functions.

The Standards define specific arguments for each Abstract Service operation. These are mapped onto corresponding arguments. Arguments that are specific to the MS Abstract Service are grouped together as a single object. For example, one argument to the **Fetch**( ) interface function is **Fetch-Argument**.

Full details of the arguments are presented in the interface function definitions of Chapter 4 and the OM class definitions of Chapter 5.

All arguments that are OM objects can generally be supplied to the interface functions as public objects (i.e., descriptor lists) or as private objects. Private objects must be created in the workspace that was returned by **Initialize**( ). In some cases, constants, representing default or commonly used instances, can be supplied instead of OM objects.

Note that wherever a function is stated to accept an instance of a particular OM class as the value of an argument, it will also accept an instance of any subclass of that OM class.

3.4.1 Attribute

Each MS attribute is represented in the interface by an OM object of the OM class **Attribute** (defined in the XDS API Specification - see reference **XDS**). The type of the MS attribute is represented by an OM attribute, **Attribute-Type**, within the OM object; the values of the MS attribute are expressed as the values of the OM attribute **Attribute-Values**. The form of each value of an MS attribute is determined by the attribute syntax associated with the type of the MS attribute (see Section 7.2 on page 126, Section 8.2 on page 134, and reference **X.413**).

The representation of the attribute value depends on the attribute type and is determined as set out below. This lists the manner in which a client program must supply values to the interface. The interface follows the same rules when returning attribute values to the client.

The attribute type and the representation of the corresponding values may be defined in the mandatory MS General Attributes Package defined in Chapter 7.

Additional attribute types and their OM representations may be defined in optional packages (e.g., the MS Interpersonal Messaging Attributes Package defined in Chapter 8, or future versions of this specification or by vendor extensions.

In the above cases, attribute values are represented as specified in the class definitions for the packages supported.

Otherwise, the attribute type is not known and the **unavailable-attribute-type** *[MS_E_UNAVAILABLE_ATTRIBUTE_TYPE]* error arises.

Where attribute values have OM syntax String(*), they may be long, segmented strings and the functions **OM-Read**( ) and **OM-Write**( ) should be used to access them.

When an attribute value is returned indicating no fields present - for example the ASN.1 contains an empty SET - then the No-Value bit of the attribute value descriptor's syntax is set, and the descriptor's value is not present.

3.4.2 AVA

An attribute value assertion (AVA) is an assertion about the value of an attribute of an (MS) entry and, in the context of MS, can be true or false. It consists of an attribute type and a single value. Loosely, the AVA is true if one of the values of the given attribute in the entry matches the given value. An AVA is represented in the interface by an instance of the OM class **AVA** (defined in the XDS API - see reference **XDS**) which is a subclass of **Attribute** constrained to have precisely one value.

3.5 Function Results

All functions return a **Status** (i.e., the function result in the C binding). Some MS functions return a **Result**. In the asynchronous mode, all functions return an **Invoke-ID**, which identifies the particular invocation. (In the C binding, the **Invoke-ID** and **Result** are returned using pointers that are supplied as arguments to the C function.) These three kinds of function results are introduced below.

All OM objects returned by interface functions (results and errors) will be private objects in the workspace returned by **Initialize**( ).

3.5.1 Invoke-ID

All interface functions that can be asynchronously invoked return an **Invoke-ID** which is an integer identifying the particular invocation of the function on a particular session. The **Invoke-ID** is only relevant for asynchronous functions and may be used later to receive the result and status. Asynchronous operations are fully described in Section 3.6 on page 28 and the interface functions that can be used to start them are indicated in Table 3-0 on page 22.

The numerical value returned from a call that successfully invokes an asynchronous operation is guaranteed to be unique among all outstanding operations within a given session. The value is such as could be returned from the Remote Operations Service Element (ROSE) defined in CCITT X.219 / X.229 and ISO 9072.

The value of the **Invoke-ID** returned for a synchronous function call is unspecified, as is that for a call that fails to invoke an operation.

3.5.2 Result

Certain functions return a result only if they succeed. For unsuccessful outcomes, errors from such functions are reported in the **Status** described below (as are errors from all other operations).

The value of **Result** returned by a function that invokes an asynchronous operation is unspecified. The result of an asynchronous operation is returned by a subsequent call to **Receive-Result**( ).

The result is returned in a private object whose OM class is appropriate to the particular function. The format of such results is driven both by the Abstract Service and by the need to provide asynchronous execution of these functions. To simplify processing of asynchronous results, the result of a single function is returned in a single OM object (corresponding to the abstract result defined in the Standards). The components of the result of a function are represented by OM attributes in the **Result** object. All information in the Abstract Service result is made available to the client. The result can be examined using functions provided in the XOM Specification - see reference **XOM**).

Any attribute values in the result are represented as described in Section 3.4.1 on page 25.

3.5.3 Status

Every interface function returns a **Status** value, which is either the constant success *{MS_SUCCESS}* or an error. Errors are represented by private objects whose OM classes are subclasses of Error, unless the interface has not been initialised succesfully in which case the constant No-Workspace *{MS_NO_WORKSPACE}* is used. Details of all errors are given in Chapter 6.

Other results of functions are not valid unless the status result has the value **success**.

3.6 Synchronous and Asynchronous Operations

The support of asynchronous operations is an optional feature; and all operations are always synchronous for implementations without such support. The support of asynchronous operations is indicated by the value of **max-outstanding-operations**, described below.

Implementations that support asynchronous operations may be executed in either the *synchronous mode* or the *asynchronous mode*; while implementations that do not can only be executed in the synchronous mode.

The asynchronous mode is chosen by appropriately setting the value of the **Maximum-Outstanding-Operations-Requested** attribute in the argument to **Initialize**( ). If this number is zero, all operations will be performed synchronously; however, if this number is greater than zero, the asynchronous mode is requested.

In *synchronous* mode, all functions wait until the operation is complete before returning. Thus, the thread of control is blocked within the interface after calling a function and it can make use of the result immediately after the function returns. (Note that in a multi-threaded system, only one thread in the process is blocked, and use of asynchronous mode is likely to be rare on such systems. On conventional single-thread process systems, the entire process is blocked; and hence the need for the asynchronous mode).

In *asynchronous* mode, some functions return before the operation is complete. The functions that can be executed asynchronously are indicated in Section 3.2 on page 22. The application is then able to continue with other processing while the operation is being performed by the Message Store; and later can access the result by calling **Receive-Result**( ). An application may initiate several concurrent asynchronous operations on the same session before receiving any of the results, subject to the limit described below. The results from asynchronously executed operations are not guaranteed to be returned in any particular order.

An asynchronous function call returns an **Invoke-ID** which uniquely identifies the function invocation. The (synchronous) function **Receive-Result**( ) returns an **Invoke-ID** corresponding to an outstanding function invocation and the results of that invocation.

Implementations will also define a limit on the number of asynchronous operations that may be outstanding at any one time on any one session. An asynchronous operation is outstanding from the time the function is called until the result is returned by **Receive-Result**( ). The limit is given by the constant

max-outstanding-operations *{MS_MAX_OUTSTANDING_OPERATIONS}*,

which can be negotiated through the **Initialize**( ) function. This limit has the value zero if asynchronous operations are not supported. If the feature is present, it is guaranteed to be at least one, so an application can always use the interface in asynchronous mode. While the maximum number of operations is outstanding, attempts to call further asynchronous

operations will report an MS Library-Error (**too-many-operations**).

A synchronous call, other than **Receive-Result**( ), may return an MS Library-Error (**mixed-synchronous**), if it is made on a session on which there are any outstanding asynchronous operations. All asynchronous operations should be allowed to terminate and their results should be obtained before making a synchronous call on the same session.

For asynchronous calls, certain forms of errors may be detected and reported immediately by the service. In such instances, the function call returns an error immediately and no outstanding operation is generated, and the value of the **Invoke-ID** is undefined. Errors detected after the asynchronous function call has returned are reported later, in the result of the function retrieved using the **Receive-Result**( ) function. All errors occurring during a synchronous request are reported when the function returns. Full details of error handling are given in Chapter 6.

Clients should ensure that there are no outstanding asynchronous operations on a session when **Unbind**( ) is called on that session. Once **Unbind**( ) has been called, there is no way to determine whether any outstanding operations have been completed. No errors or results of any kind will be reported to the client and outstanding calls may be left partially completed. Hence, it is strongly recommended that before closing a session, **Receive-Result**( ) be called repeatedly until the **Completion-Flag** takes the value of **no-outstanding-operation**.

3.7 Security

It is not the purpose of this interface specification to constrain the security policy of any implementation or local administration. Such policies may differ widely according to the requirements of different user groups.

Chapter 4

Interface Functions

This chapter defines the MS interface functions. It specifies the functions that the service makes available to the client, the data types used by the interface functions as well as the functional units available for the client to request the usage of subsets of MS functions.

4.1 Data Types

This section defines, and the following table lists, the data types of the MS interface. The data types of both the generic and C interfaces are specified. The interface also uses other data types, e.g., Boolean, Object, Object Identifier, Private Object, String and intermediate data types of the OM interface (see XOM Specification in **Referenced Documents**).

Data Type	Description
Feature	lists the features requested for a session
Invoke-ID	identifies a particular invocation of an interface function in the asynchronous mode
Status	indicates whether a function has succeeded or not; and if not, gives the possible error value

Table 4-1 Interface Data Types

NAME

Feature - type definition for requesting features for a session

SYNOPSIS

```
#include <xms.h>

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

DESCRIPTION

A data value of this type is used for requesting the features on a session. See Table 1-1 on page 5 for the object identifiers identifying the features defined in this specification.

NAME

Invoke-ID - type definition for identifying a particular interface function invocation in the asynchronous mode

SYNOPSIS

```
#include <xms.h>

typedef OM_sint           MS_invoke_id;
```

DESCRIPTION

A data value of this type is used for identifying a particular interface function invocation in the asynchronous mode

NAME

Status - type definition for indicating the outcome (success or error) of a function

SYNOPSIS

```
#include <xms.h>

typedef OM_private_object          MS_status;
```

DESCRIPTION

A data value of this type is used for indicating the outcome (success or error) of a function. For further information, refer to Section 3.5.3 on page 28.

4.2 Functional Units

A client program may request certain features of the Message Store it would like to use for the duration of an API session. Such features may be negotiated in terms of functional units and packages. A *functional unit* (FU) is a collection of related functions. The following table shows the interface functions available within each functional unit.

The MS FU is mandatory.

MS	MS Submission	MS Administration	MS Alert
Bind Delete Fetch Initialize List Receive-Result † Register-MS Shutdown Summarize Unbind Wait	Cancel-Submission Submit	Register	Check-Alert

† available only if asynchronous mode is supported

Table 4-2 Functional Units

Functional units are requested using the **Feature-List** argument to the **Initialize**( ) function. After the interface has been initialised with certain features negotiated, if the client attempts to invoke functions not previously negotiated, an error,

feature-not-requested *[MS_E_FEATURE_NOT_REQUESTED],*

occurs.

All FUs must be used together with the MS General Attributes Package (see Chapter 7). Note that the functions of the Submission FU and the Administration FU require class definitions to be imported from the Message Handling Package of X.400 API Specification (see reference **X.400**).

4.3 Function Definitions

The following are the definitions of the interface functions along with their C bindings.

All errors, including Message Store errors (see reference **X.413**), are returned in **Status** (see Section 3.5.3 on page 28.

NAME

Bind - establish a session with the Message Store

SYNOPSIS

```
#include <xms.h>

MS_status  ms_bind (
    OM_object                   bind_argument,
    OM_private_object          *bind_result_return,
    OM_private_object          *bound_session_return
);
```

DESCRIPTION

This function establishes a session with the Message Store. It must be called after **Initialize**( ) and before any other Message Store interface functions are called.

ARGUMENTS

1. **Bind-Argument** (Object(Bind-Argument))
   specifies information for establishing a session with the Message Store service provider, together with details of the service required. This comprises:

   — **Initiator**
   specifies the OR-name of the initiator (i.e., the UA) of this session (or association) with the MS.

   — **Initiator-Credentials**
   specifies the credentials of the initiator for authentication purposes.

   — **Security-Context**
   identifies the security context at which the initiator proposes to operate.

   — **Fetch-Restrictions**
   specifies the restrictions on entries to be returned as result of a **Fetch**( ) function. These restrictions prevail until the **Unbind**( ) function is invoked.

   — **MS-Configuration-Request**
   if true, specifies the request to obtain information relating to which auto-actions and optional attributes the MS provides support for. If false, no such request is being made.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not.

2. **Bind-Result** (Object(Bind-Result))
   on successful completion, contains one or more of the following:

   — **Responder-Credentials**
   specifies the credentials of the MS.

   — **Available-Auto-Actions**
   specifies the set of all possible auto-actions that are supported by the MS (not just those requested by the client), if the MS-Configuration-Request was made for the **Bind**( ) function.

   — **Available-Attribute-Types**
   specifies the set of all optional MS attribute-types that are supported by the MS, if the MS-Configuration-Request was made for the **Bind**( ) function.

— **Alert-Indication**
indicates an alert condition has occurred since the last successful Alert-indication.

— **Content-Types-Supported**
specifies a set of object-identifiers defining the content-types of which the MS has knowledge, if the MS-Configuration-Request was made for the **Bind**( ) function.

3. **Bound-Session** (Object(Session))
upon successful completion, contains an instance of the Session class describing an association between the client and the service. This value is then used as an input argument, **Session**, to other functions (e.g., **Fetch**( )).

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, too-many-sessions, miscellaneous.

The following Message Store errors may be returned:
authentication-error, no-workspace, unacceptable-security-context, unable-to-establish-association.

This function can return a **Communications-Error**.

SEE ALSO

Unbind( ).

NAME

Cancel-Submission - cancel a message submitted with the deferred delivery option

SYNOPSIS

```
#include <xms.h>

MS_status  ms_cancel_submission (
    OM_private_object           session,
    OM_object                   mts_identifier,
    MS_invoke_id               *invoke_id_return
);
```

DESCRIPTION

This function attempts to cancel the delivery of a message submitted with the deferred delivery option, regardless of the session in which it was submitted.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **MTS-Identifier** (Object(MTS-Identifier))
   refers to the MTS-Identifier assigned to the messages whose delivery is to be cancelled.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Invoke-ID** (Integer)
   specifies the Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, bad-class, no-such-class, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
deferred-delivery-cancellation-rejected, message-submission-identifier-invalid, no-workspace, remote-bind-error.

This function can return a **Communications-Error**.

SEE ALSO

Submit( ).

NAME

Check-Alert - check if the MS has received new entries whose attributes match the criteria previously supplied by the **Register-MS**( ) function

SYNOPSIS

```
#include <xms.h>

MS_status  ms_check_alert (
    OM_private_object              session,
    OM_private_object             *check_alert_result_return,
);
```

DESCRIPTION

This function is used to check if the MS has received new entries whose attributes match the criteria previously supplied by the **Register-MS**( ) function.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not.
2. **Result** (Object(Check-Alert-Result))
   if there are alerts, each element of the result may contain the following:
   - **Alert-Registration-Identifier**
     identifies which of the auto alert registrations resulted in the alert.
   - **New-Entry**
     if present, conveys the information from the new entry which was requested in the auto alert registration parameter; otherwise, is absent when the user did not specify an auto alert registration parameter.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-session, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too many operations.

The following Message Store errors may be returned:
no-workspace, security-error.

This function can return a **Communications-Error**.

SEE ALSO

Register-MS( ), **Wait**( ).

NAME

Delete - remove selected entries from an information base

SYNOPSIS

```
#include <xms.h>

MS_status  ms_delete (
    OM_private_object          session,
    OM_object                  delete_argument,
    MS_invoke_id              *invoke_id_return
);
```

DESCRIPTION

This function is used to delete selected entries from an information base. A main-entry and all its dependent child-entries may only be deleted together. This is achieved by specifying just the main-entry as an argument. The function will only be successful when operating on those information-bases permitted by the security-context and the security-policy in force.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **Delete-Argument** (Object(Delete-Argument))
   specifies the argument for the **Delete**( ) function. It comprises:
   - **Information-Base-Type**
     specifies which information base type is being addressed (see Section 6.4.1 in reference **X.413**). Its value must be one of the following:
     - **stored-messages** *{MS_STORED_MESSAGES}*
       specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**).
     - **inlog** *{MS_INLOG}*
     - **outlog** *{MS_OUTLOG}*
   - **Items**
     specifies the entries to be deleted.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Invoke-ID** (Integer)
   specifies the Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, bad-class, no-such-class, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
delete-error, invalid-parameter-error, no-workspace, range-error, security-error, sequence-number-error, service-error.

This function can return a **Communications-Error**.

NAME

Fetch - get selected information from a specific entry in an information base

SYNOPSIS

```
#include <xms.h>

MS_status  ms_fetch (
    OM_private_object            session,
    OM_object                    fetch_argument,
    OM_private_object           *fetch_result_return,
    MS_invoke_id                *invoke_id_return
);
```

DESCRIPTION

This function is used to obtain selected information from a specific entry in an information base; alternatively, it is used to obtain the first entry from among several entries of interest, in which case, the sequence numbers of the other selected entries are also returned.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **Fetch-Argument** (Object(Fetch-Argument))
   specifies the argument for the **Fetch**( ) function. It comprises:
   - **Information-Base-Type**
     specifies which information base type is being addressed (see Section 6.3.1 in reference **X.413**.) Its value must be one of the following:
     - **stored-messages** *{MS_STORED_MESSAGES}*
       specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**).
     - **inlog** *{MS_INLOG}*
     - **outlog** *{MS_OUTLOG}*
   - **Item**
     specifies the selector determining the entry to be fetched.
   - **Requested-Attributes**
     indicates what information from the selected entry is to be returned in the result.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Result** (Object(Fetch-Result))
   on successful completion of a synchronous call, contains one or more of the following:
   - **Entry-Information**
     specifies the requested entry-information from the selected entry (unless the function returns without any entries being selected).

— **List**
specifies, in the case a search was performed and more than one entry matching the selector were found, a list of sequence-numbers, in ascending order, of these entries.

— **Next**
specifies, in the case where the number of entries selected would have been greater if it were not for the limit specified in the selector, the sequence-number for the next entry.

3. **Invoke-ID** (Integer)
specifies the Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, bad-class, no-such-class, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
attribute-error, fetch-restriction-error, invalid-parameter-error, no-workspace, range-error, security-error, sequence-number-error, service-error.

This function can return a **Communications-Error**.

NAME

Initialize - initialise the interface, returning a workspace.

SYNOPSIS

```
#include <xms.h>

MS_status  ms_initialize (
    MS_feature                feature_list[],
    OM_sint                  *max_sessions,
    OM_sint                  *max_outstanding_operations,
    OM_workspace             *workspace_return,
);
```

DESCRIPTION

This function performs any necessary initialisation of the interface, including creating a workspace and making the mandatory features (see Section 1.4 on page 5) available. It must be called before any other Message Store interface functions are called. If it is subsequently called before **Shutdown**( ), the result is unspecified.

ARGUMENTS

1. **Feature-List** (Feature-List)
   identifies the additional features (FUs and packages) requested by the client. This is an ordered sequence of features, each represented by an object identifier (see Section 1.4 on page 5 for the features defined in this specification). The sequence is terminated by an object identifier having no components (a length of zero and any value of the data pointer in the C representation). Mandatory features are made available even if no features are requested.
2. **Maximum-Sessions-Requested** (Integer)
   indicates the maximum number of simultaneous MS sessions requested by the client.
3. **Maximum-Outstanding-Operations-Requested** (Integer)
   indicates the maximum number of outstanding asynchronous operations requested by the client.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not.
2. **Activated** (Boolean-List)
   if the function completed successfully, contains an ordered sequence of Boolean values, with the same number of elements as the **Feature-List**. A value of true indicates that the corresponding feature is part of the interface whereas a value of **false** indicates that the corresponding feature is not available.

In the C binding, this result is combined with the **Feature-List** argument as a single array of structures of type *MS_feature*, which is defined as:

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

3. **Maximum-Sessions-In-Effect** (Integer)
   gives the maximum number of simultaneous MS sessions requested or the maximum number that can be supported by the service, whichever value is lesser.

   In the C binding, the **Maximum-Sessions-Requested** argument and the **Maximum-Sessions-In-Effect** result of the generic interface are realised as the **max_sessions** argument.

4. **Maximum-Outstanding-Operations-In-Effect** (Integer)
   gives the maximum number of outstanding asynchronous operations or the maximum number that can be supported by the service, whichever value is lesser. If the service does not support asynchronous operations, then the result returned is zero; otherwise it is a positive integer.

   In the C binding, the **Maximum-Outstanding-Operations-Requested** argument and the **Maximum-Outstanding-Operations-In-Effect** result of the generic interface are realised as the **max_outstanding_operations** argument.

5. **Workspace** (Workspace)
   on successful completion, contains a handle to a workspace in which OM objects can be created and manipulated. Objects created in this workspace may be used as arguments to other interface functions.

ERRORS

This function can only return a **No-Workspace** error.

SEE ALSO

Shutdown( ).

NAME

List - return selected information for a list of entries of interest‘ from an information base

SYNOPSIS

```
#include <xms.h>

MS_status  ms_list (
    OM_private_object           session,
    OM_object                   list_argument,
    OM_private_object          *list_result_return,
    MS_invoke_id               *invoke_id_return
);
```

DESCRIPTION

This function is used to obtain selected information for a list of entries selected from an information base.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **List-Argument** (Object(List-Argument))
   specifies the argument for the *List*( ) function. It comprises:
   - **Information-Base-Type**
     specifies which information base type is being addressed (see Section 6.3.1 in reference **X.413**.) Its value must be one of the following:
     - **stored-messages** *{MS_STORED_MESSAGES}*
       This type specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**).
     - **inlog** *{MS_INLOG}*
     - **outlog** *{MS_OUTLOG}*
   - **Selector**
     determines the entries are to be returned.
   - **Requested-Attributes**
     indicates the information from the selected entry to be returned in the result.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Result** (Object(List-Result))
   on successful completion of a synchronous call, contains one or both of the following:
   - **Next**
     specifies the sequence-number of the next entry that would have been selected in the case where the number of entries selected would have been greater if it were not for the limit specified in the **Selector**.

— **Requested**
specifies the requested entry-information from each selected entry (one or more), in ascending order of sequence-number (unless the function returns without any entries being selected).

3. **Invoke-ID** (Integer)
The Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
attribute-error, invalid-parameter-error, no-workspace, range-error, security-error, sequence-number-error, service-error.

This function can return a **Communications-Error**.

NAME

Receive-Result - retrieve the result of an asynchronously executed operation

SYNOPSIS

```
#include <xms.h>

MS_status  ms_receive_result (
    OM_private_object           session,
    OM_uint                    *completion_flag_return,
    MS_status                  *operation_status_return,
    OM_private_object          *result_return,
    MS_invoke_id               *invoke_id_return
);
```

DESCRIPTION

This function is used to retrieve the completed result of some operation previously executed asynchronously.

The function results include two status indications. One, called, **Status**, indicates whether this function call itself was successful; it is always returned. The other, called **Operation-Status**, is used to return the status of the completed asynchronous operation and is only returned if it exists.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not.
2. **Completion-Flag** (Unsigned-Integer)
   indicates the status of outstanding asynchronous operations and has one of the following values:

   — **completed-operation** *{MS_COMPLETED_OPERATION}*
   indicates that at least one outstanding asynchronous operation has completed and the result for one is made available.

   — **outstanding-operation** *{MS_OUTSTANDING_OPERATION}*
   indicates that there are outstanding asynchronous operations, but none has yet completed.

   — **no-outstanding-operation** *{MS_NO_OUTSTANDING_OPERATION}*
   indicates that there are no outstanding asynchronous operations.

   This result is valid if **Status** has the value **success**.

   On successful return with **Completion-Flag** having the value **completed-operation**, the Status and the Invoke-ID of the completed operation are returned.
3. **Operation-Status** (Status)
   indicates whether the asynchronous operation succeeded or not; if not, the possible error values are those listed for the individual operation in the corresponding function description.

   This result is valid if **Status** has the value success and **Completion-Flag** has the value **completed-operation**.

4. **Result** (Object(*))
   gives the result of the completed asynchronous operation. Its value will be the constant **Null-Result** *{MS_NULL_RESULT}* if the operation was one that does not return a result. Otherwise, the OM object's OM class is that of the result of the asynchronous operation and can be determined using the OM functions.

   Note: the possible forms of ''result_return'' that **Receive-Result**( ) is required to support is restricted to the results of interface functions within the FUs supported.

   This result is valid if **Status** has the value success and **Completion-Flag** has the value **completed-operation**.

5. **Invoke-ID** (Integer)
   specifies the Invoke-ID of the completed asynchronous operation whose result is being returned. This result is valid if **Status** has the value success and **Completion-Flag** has the value **completed-operation**.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-session, feature-unavailable, feature-not-negotiated, no-workspace, out-of-memory, miscellaneous.

This function does not report any Message Store errors or a **Communications-Error** in its **Status** result. (Any such errors related to the completed asynchronous operation are reported in **Operation-Status**, as described above.)

SEE ALSO

All interface functions that can be asynchronously executed.

NAME

Register - modify various parameters held by the MTS regarding delivery of messages to the MS

SYNOPSIS

```
#include <xms.h>

MS_status  ms_register (
    OM_private_object              session,
    OM_object                      register_argument,
    MS_invoke_id                  *invoke_id_return
);
```

DESCRIPTION

This function is used to make long-term modifications to various parameters held by the MTS regarding the delivery of messages to the MS.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.

2. **Register-Argument** (Object(Register-Argument))
   specifies how to modify (via the **Register**( ) function) various parameters held by the MTS regarding delivery of messages to the MS. These parameters include the following:

   — **User-Name**
   specifies the OR-name of the MS (this corresponds to the OR-name of the UA), if the user-name is to be changed.

   — **User-Address**
   specifies the OR-address of the MS (this corresponds to the OR-name of the UA), if it is required by the MTS and if it is to be changed.

   — **Deliverable-Encoded-Information-Types**
   indicates the encoded-information-types that the MTS shall permit to appear in messages delivered to the MS, if they are to be changed. The MTS shall reject as undeliverable any message for an MS for which the MS is not registered to accept delivery of all of the encoded-information-types of the message. Note that the MS may register to receive the undefined encoded-information-type. This argument, **Deliverable-Encoded-Information-Types**, also indicates the possible encoded-information-types to which implicit conversion can be performed.

   — **Deliverable-Maximum-Content-Length**
   indicates the content-length, in octets, of the longest content message that the MTS shall permit to appear in messages being delivered to the MS, if it is to be changed. The MTS shall reject as undeliverable any message for an MS for which the MS is not registered to accept delivery of messages of its size.

— **Default-Delivery-Controls**
indicates default delivery controls which are registered using the **Register**( ) function. The default delivery control arguments shall not admit messages whose delivery are prohibited by the prevailing registered values of the **Deliverable-Encoded-Information-Types** argument, the **Deliverable-Content-Types** argument or the **Deliverable-Maximum-Content-Length** argument.

— **Deliverable-Content-Types**
indicates the content-types that the MTS shall permit to appear in messages delivered to the MS, if they are to be changed. The MTS shall reject as undeliverable any message for an MS for which the MS is not registered to accept delivery of all of the content-types of the message. Note that the MS may register to receive the **undefined** content-type.

— **Labels-And-Redirections**
contains one or both of the following:

— the OR-name of an alternate recipient to which messages are to be redirected, if this is to be changed.

— the security-labels of the UA, if they are to be changed.

RESULTS

1. **Status** (Status)
Whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.

2. **Invoke-ID** (Integer)
The Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, no-workspace, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
register-rejected.

This function can return a **Communications-Error**.

NAME

Register-MS - register or deregister various information with the MS

SYNOPSIS

```
#include <xms.h>

MS_status  ms_register_ms(
    OM_private_object               session,
    OM_object                       register_ms_argument,
    MS_invoke_id                   *invoke_id_return
);
```

DESCRIPTION

This function is used to register or deregister various information with the MS.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **Register-MS-Argument** (Object(Register-MS-Argument))
   specifies the argument for the **Register-MS**( ) function. It comprises:
   - **Auto-Action-Registrations**
     specifies a set of auto-action-registrations, one for each auto-action to be registered.
   - **Auto-Action-Deregistrations**
     specifies a set of auto-action-deregistrations, one for each auto-action to be deregistered.
   - **List-Attribute-Defaults**
     specifies a default set of attribute-types to indicate which attributes should be returned for any subsequent **List**( ) function if the entry-information-selection argument is absent. This value replaces any previously registered default set. If absent, no change will be applied to the registered default set.
   - **Fetch-Attribute-Defaults**
     specifies a default set of attribute-types to indicate which attributes should be returned for any subsequent **Fetch**( ) function if the entry-information-selection argument is absent. This value replaces any previously registered default set. If absent, no change will be applied to the registered default set.
   - **Change-Credentials**
     specifies the old and new credentials of the end user, if change credentials was requested.
   - **User-Security-Labels**
     contains the security-labels of the UA, if they are to be changed.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Invoke-ID** (Integer)
   specifies the Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, bad-class, no-such-class, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
attribute-error, auto-action-request-error, invalid-parameter-error, no-workspace, security-error, service-error.

This function can return a **Communications-Error**.

NAME

Shutdown - shuts down the interface, discarding the workspace

SYNOPSIS

```
#include <xms.h>

void  ms_shutdown (
    void
);
```

DESCRIPTION

This function shuts down the interface established by **Initialize**( ), and may enable the service to release resources (in its workspace). All sessions will become invalid and no other MS interface functions should be called after this function except for **Initialize**( ). This function does not return any results or errors.

ARGUMENTS

None.

RESULTS

None.

ERRORS

None.

SEE ALSO

Initialize( ).

NAME

Submit - submit a communique (message or probe)

SYNOPSIS

```
#include <xms.h>

MS_status  ms_submit (
    OM_private_object           session,
    OM_object                   communique,
    OM_private_object          *submission_results_return,
    MS_invoke_id               *invoke_id_return
);
```

DESCRIPTION

After verifying the integrity of the communique, this function submits a communique (message or probe) by adding it to the submission queue to which the current session provides access. A message may be submitted by requesting the forwarding of a delivered message identified by its sequence-number.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **Communique** (Object(Submitted-Communique))
   specifies the object (a message or a probe) to be submitted. Its purported originator shall be the user associated with the session. If the communique is a private object, it is made inaccessible to the client; and is deleted at the discretion of the service.

   Note: If the subclass, **Item-To-Forward**, were used for this argument, it is possible to request forwarding a delivered message identified by its MS sequence-number. (The MS entry to be thus forwarded should be a delivered message entry. Forwarding of entries that are not delivered messages is not defined in this specification.)

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Result** (Object(Submission-Results))
   on successful completion of a synchronous call, contains the results of the submission. For details, refer to the X.400 API (see reference **X.400**).
3. **Invoke-ID** (Integer)
   specifies the Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, bad-class, no-such-class, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
sequence-number-error, submission-control-violated, element-of-service-not-subscribed, no-workspace, originator-invalid, recipient-improperly-specified, inconsistent-request, security-error, unsupported-critical-function, remote-bind-error.

This function can return a **Communications-Error**.

SEE ALSO

Cancel-Submission( ).

NAME

Summarize - summarise counts of selected entries in an information base

SYNOPSIS

```
#include <xms.h>

MS_status  ms_summarize (
    OM_private_object                session,
    OM_object                        summarize_argument,
    OM_private_object               *summarize_result_return,
    MS_invoke_id                    *invoke_id_return
);
```

DESCRIPTION

This function is used to obtain summary counts of selected entries in an information base; in addition, a count of entries selected and their lowest and highest sequence-numbers are also returned. Zero or more individual summaries may be requested.

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **Summarize-Argument** (Object(Summarize-Argument))
   specifies the argument for the **Summarize**( ) function. It comprises:
   - **Information-Base-Type**
     specifies which information base type is being addressed (see Section 6.3.1 in reference **X.413**). Its value must be one of the following:
     - **stored-messages** *{MS_STORED_MESSAGES}*
       specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**).
     - **inlog** *{MS_INLOG}*
     - **outlog** *{MS_OUTLOG}*
   - **Selector**
     specifies the set of criteria for determining the entries which are to be summarised.
   - **Summary-Requests**
     indicates the sequence of Attribute-Types for which summaries are requested.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Result** (Object(Summary-Result))
   on successful completion of a synchronous call, contains one or more of the following:
   - **Next**
     is returned in the case where the number of entries selected would have been greater if it were not for the limit specified in the selector. This would then give the sequence-number for the next entry that would have been selected.

— **Count**
gives the number of entries that matched the selection criteria.

— **Span**
gives the range of sequence-numbers of entries that matched the selection criteria. It is absent if there were no such entries (i.e., **Count** value is zero).

— **Summaries**
is a sequential list of summaries; one for each summary-request. The summaries are returned in the order that they were requested in the Summary function.

3. **Invoke-ID** (Integer)
specifies the Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, bad-class, no-such-class, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, out-of-memory, miscellaneous, too-many-operations.

The following Message Store errors may be returned:
attribute-error, invalid-parameter-error, no-workspace, range-error, security-error, sequence-number-error, service-error.

This function can return a **Communications-Error**.

NAME

Unbind - terminate a session with the Message Store

SYNOPSIS

```
#include <xms.h>

MS_status  ms_unbind (
    OM_private_object          session
);
```

DESCRIPTION

This function terminates a session with the Message Store.

Note: This implies that the results of any outstanding asynchronous operations initiated in this session can no longer be received and it is not possible to know whether or not such operations succeeded. Any such operations may have been carried out or terminated prematurely. For this reason, it is recommended that all outstanding asynchronous operations be processed using **Receive-Result**( ) before calling **Unbind**( ).

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session to be unbound.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**: bad-session, no-workspace, out-of-memory, miscellaneous.

This function can return a **Communications-Error**.

SEE ALSO

Bind( ).

NAME

Wait - return when a new entry is available in the Message Store for retrieval or when a period of time has elapsed, whichever occurs first

SYNOPSIS

```
#include <xms.h>

MS_status  ms_wait (
    OM_private_object           session,
    OM_uint32                   interval,
    OM_private_object          *wait_result_return,
    MS_invoke_id               *invoke_id_return
);
```

DESCRIPTION

This function returns when a new entry is available in the Message Store for retrieval or when a period of time has elapsed, whichever occurs first. (see reference **X.413** for the definition of *new*, a possible value for the Entry-Status MS attribute.)

ARGUMENTS

1. **Session** (Object(Session))
   specifies the MS session over which this function is performed.
2. **Interval** (Integer)
   specifies the maximum length of time (in milliseconds) that the service is to wait before returning a result when checking for entries in the Message Store available for retrieval.

RESULTS

1. **Status** (Status)
   indicates whether the function succeeded or not, if used synchronously; or whether the function has been initiated, if used asynchronously.
2. **Result** (Object(Wait-Result))
   on successful completion of a synchronous call, will be either:
   - — the constant **No-New-Entries** *{MS_NO_NEW_ENTRIES}* if there are no new MS entries available for retrieval.
   - — a private object if there are new MS entries available for retrieval.
3. **Invoke-ID** (Integer)
   specifies the Invoke-ID of the asynchronous operation.

ERRORS

This function can return a **System-Error** or one of the following **Library-Errors**:
bad-argument, bad-session, asynchrony-not-supported, feature-unavailable, feature-not-negotiated, no-workspace, out-of-memory, miscellaneous, too-many-operations.

This function can return a **Communications-Error**.

Chapter 5

Interface Class Definitions

5.1 Introduction

This chapter defines, in alphabetical order, the OM classes that constitute the Message Store Package. The errors defined in the next chapter also belong to this package. The Object Identifier associated with this package is represented by the constant **MS-Package** *{MS_PACKAGE}*. See Table 1-1 on page 5 for the value of this object identifier.

Note that the possible forms of ‘‘result_return’’ that **Receive-Result**( ) is required to support is restricted to the results of interface functions within the FUs supported.

The concepts of Object Management were briefly described in Section 1.6 on page 9, and the notation is introduced below. Full details are given in the XOM Specification (see reference **XOM**).

Each OM class is described in a separate section which identifies the OM attributes specific to that OM class. The OM classes and OM attributes for each OM class are listed in alphabetical order. The OM attributes that may be found in an instance of an OM class are those OM attributes specific to that OM class as well as those inherited from each of its superclasses.

The OM class-specific OM attributes are defined in a table. The table gives the name of each OM attribute, the syntax of each of its values, any restrictions upon the length (in bits, octets / bytes or characters) of each value, any restrictions upon the number of values and the value, if any, the **OM-Create**( ) function supplies.

The constants that represent the OM classes and OM attributes in the C binding are defined in the **<xms.h>** header (see Chapter 9).

5.2 Class Hierarchy

This section depicts the hierarchical organisation of the classes defined in this chapter and thus shows which classes inherit additional OM attributes from their superclasses. Subclassification is indicated by indentation and the names of abstract classes are rendered in italics. Thus, for instance, the concrete class **Fetch-Attribute-Defaults** is an immediate subclass of the abstract class *Attribute-Defaults* which in turn is an immediate subclass of the abstract class *Object*.

The following symbols denote FUs in which a class is used.

Symbol	FU in which a class is used
[M]	MS FU
[S]	Submission FU
[A]	Administration FU
[C]	Alert FU

Object	(defined in XOM Spec.)	[M]	[S]	[A]	[C]
- Address	(defined in XDS Spec.)			[A]	
- Alert-Address		[M]			
- Attribute	(defined in XDS Spec.)		[M]		
- AVA	(defined in XDS Spec.)	[M]			
- Filter-Item	(defined in XDS Spec.)	[M]			
- *Attribute-Defaults*		[M]			
- Fetch-Attribute-Defaults		[M]			
- List-Attribute-Defaults		[M]			
- Attribute-Selection		[M]			
- *Auto-Action*		[M]			
- Auto-Action-Deregistration		[M]			
- Auto-Action-Registration		[M]			
- Auto-Alert-Registration-Parameter		[M]			
- Auto-Forward-Registration-Parameter		[M]			
- Bind-Argument		[M]			
- Bind-Result		[M]			
- Change-Credentials		[M]			
- Check-Alert-Result					[C]
- *Common-Controls*				[A]	
- Default-Delivery-Controls				[A]	
- Creation-Time-Range		[M]			
- Credentials		[M]			
- Delete-Argument		[M]			
- Deliverable-Content-Types				[A]	
- EITs	(defined in X.400 Spec.)			[A]	
- *Error*	(see Chapter 6)	[M]			
- Fetch-Argument		[M]			
- Fetch-Result		[M]			
- Filter	(defined in XDS Spec.)	[M]			
- Items		[M]			
- Item		[M]			
- Label-And-Redirection				[A]	
- Labels-And-Redirections				[A]	
- List-Argument		[M]			
- List-Result		[M]			

-	OR-Name	defined in X.400 Spec.	[M]	[S]	[A]
-	MS-Entry-Information		[M]		
-	MS-Entry-Information-Selection		[M]		
-	MTS-Identifier	(defined in X.400 Spec.)		[S]	
-	Password		[M]		
-	Range		[M]		
-	Register-Argument				[A]
-	Register-MS-Argument		[M]		
-	Restrictions		[M]		
-	Security-Label	(defined in [X.400 Spec.])	[M]		
-	Selector		[M]		
-	Sequence-Number-Range		[M]		
-	Session		[M]		
-	Strong-Credentials		[M]		
-	*Submitted-Communique*	(defined in X.400 Spec.)		[S]	
	- Submitted Message	(defined in X.400 Spec.)		[S]	
	- Item-to-Forward			[S]	
	- Auto-Forward-Arguments			[S]	
	- Submitted Probe	(defined in X.400 Spec.)		[S]	
-	Submission-Results	(defined in X.400 Spec.)		[S]	
-	Summarize-Argument		[M]		
-	Summary		[M]		
-	Summary-Present		[M]		
-	Summary-Requests		[M]		
-	Summary-Result		[M]		
-	Wait-Result		[M]		

The client is not permitted to create or modify instances of certain OM classes because these OM classes are only returned by the interface and never supplied to it. Such OM classes are:

- all subclasses of Error
- Bind-Result
- Check-Alert-Result
- Fetch-Result
- List-Result
- MS-Entry-Information
- Submission-Results
- Summary
- Summary-Present
- Summary-Result
- Wait-Result

Note that the terms ‘‘Entry-Information’’ and ‘‘Entry-Information-Selection’’ are used in both X.500 and the MS protocols but they have different ASN.1 syntax definitions. Hence, we make the distinction by using ‘‘MS-Entry-Information’’ and ‘‘MS-Entry-Information-Selection’’ in this MS specification document.

5.3 Address

As defined in the Directory Services Package in the XDS Specification (see reference **XDS**).

5.4 Alert-Address

The OM class **Alert-Address** gives the alert information to used by the auto-alert operation.

An instance of OM class **Alert-Address** has the OM attributes of its superclass: *Object* and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Address	Object(External)	-	1	-
Alert-Qualifier	String(Octet)	-	0-1	-

Table 5-1 OM Attributes of an Alert-Address

Address
Identifies the type of alert address to be invoked.

Alert-Qualifier
Contains any further information which need to be included with the auto-alert.

5.5 Attribute

As defined in the Directory Services Package in the XDS Specification (see reference **XDS**).

5.6 Attribute-Defaults

The OM class **Attribute-Defaults** specifies a default set of attribute-types to indicate which attributes should be returned for any subsequent List or Fetch functions if the Entry-Information-Selection argument is absent. If absent, no change will be applied to the registered default.

The OM class **Attribute-Defaults** is an abstract class and has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attribute-Type	String(Object-Identifier)	-	0/more	-

Table 5-2 OM Attributes of Attribute-Defaults

Attribute-Type
This specifies the default set of attribute-types to indicate which attributes should be returned for any subsequent List or Fetch functions if the Requested-Attributes OM attribute of List-Argument or Fetch-Argument respectively were absent. If absent, no change will be applied to the registered default.

5.7 Attribute-Selection

An instance of OM class **Attribute-Selection** describes the selection of attributes of an entry being requested.

An instance of OM class **Attribute-Selection** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attribute-Type	String(Object-Identifier)	-	1	-
From	Integer	-	0-1	-
Count	Integer	-	0-1	-

Table 5-3 OM Attributes of Attribute-Selection

Attribute-Type

This indicates the class of information given by this MS attribute.

From

This may only be present if the attribute-type is multi-valued. When an MS attribute is multi-valued, this indicates the relative position of the first value to be returned. If it specifies a value beyond those present in the MS attribute, no values are returned. If it is omitted, values starting at the first value are returned.

Note that MS attributes are numbered starting at position 1.

Count

This may only be present if the attribute-type is multi-valued. When an MS attribute is multi-valued, this gives the number of values to be returned. If there are less than count values present in the MS attribute, all values are returned. If it is omitted, there is no limit as to how many values are returned.

5.8 AVA

As defined in the Directory Services Package in the XDS Specification (see reference **XDS**).

5.9 Auto-Action

The OM class **Auto-Action** describes an action that will occur automatically whenever the associated registration criteria have been satisfied. The result of an action being invoked is visible externally to the MS. Auto-actions are registered with the MS using the **Register-MS**( ) function.

The OM class **Auto-Action** is an abstract class which has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Type	String(Object-Identifier)	-	1	-
Registration-ID	Integer	-	1	-

Table 5-4 OM Attributes of Auto-Action

Type
This indicates the class of auto-action type.

Registration-ID
This identifies a particular auto-action registration.

5.10 Auto-Action-Deregistration

An instance of OM class **Auto-Action-Deregistration** describes an auto-action-registration to be deregistered using the **Register-MS**( ) function. Any auto-action with registration-identifier and auto-action-type matching that specified is deregistered.

An instance of the OM class **Auto-Action-Deregistration** has the OM attributes of its superclasses: *Object*, **Auto-Action** and no additional OM attributes.

5.11 Auto-Action-Registration

An instance of OM class **Auto-Action-Registration** describes an auto-action-registration to be registered using the **Register-MS**( ) function. The new auto-action-registration-parameter supersedes any previously registered auto-action (if any) with that registration-identifier and auto-action-type.

An instance of the OM class **Auto-Action-Registration** has the OM attributes of its superclasses: *Object*, **Auto-Action** and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Registration-Parameter	Any	-	1	-

Table 5-5 OM Attributes of Auto-Action-Registration

Registration-Parameter
This specifies the object identifier of an auto-action type to which an auto-action registration must conform.

5.12 Auto-Alert-Registration-Parameter

An instance of OM class **Auto-Alert-Registration-Parameter** specifies the criteria to determine whether a user should be alerted to the delivery of a message into the stored message information base.

An instance of the OM class **Auto-Alert-Registration-Parameter** has the OM attributes of its superclass: *Object*, and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Filter	Object(Filter)	-	0-1	-
Alert-Address	Object(Alert-Address)	-1/more	-	
Requested-Attributes	Object(MS EntryInformationSelection)	-	0-1	-

Table 5-6 OM Attributes of Auto-Alert-Registration-Parameter

Filter
The set of criteria which a delivered message must satisfy for the auto alert function to be activated with the given set of parameters.

Alert-Address
The types of alert services to be invoked.

Requested-Attributes
This indicates what information from the selected entries is to be included with the auto-alert.

5.13 Auto-Forward-Arguments

An instance of OM class **Auto-Forward-Arguments** describes the set of criteria to be used in the auto-forwarding of a submitted message.

An instance of the OM class **Auto-Forward-Arguments** has the OM attributes of its superclass: *Object*, and additionally the attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Alternate-Recipient-Allowed	Boolean	-	1	false
Confidentiality-Algorithm	Object (Algorithm)	-	0-1	-
Content-Correlator	*any*	3-512	0-1	-
Content-Identifier	String (Printable)	1-16	0-1	-
Content-Return-Requested	Boolean	-	1	false
Conversion-Loss-Prohibited	Boolean	-	1	false
Conversion-Prohibited	Boolean	-	1	false
Deferred-Delivery-Time	String (UTC Time)	0-17	0-1	-
Disclosure-Allowed	Boolean	-	1	false
Expansion-Prohibited	Boolean	-	1	false
Latest-Delivery-Time	String (UTC Time)	0-17	0-1	-
Origin-Check	Object (Algorithm and Result)	-	0-1	-
Original-EITs	Object (EITs)	-	0-1	-
Originator-Certificate	Object (Certificates)[1]	-	0-1	-
Originator-Name	Object (OR Name)	-	1	-
Originator-Return-Address	Object (OR Address)	-	0-1	-
Priority	Enum (Priority)	-	1	normal
Proof-of-Submission-Requested	Boolean	-	1	false
Reassignment-Prohibited	Boolean	-	1	false
Recipient-Descriptors	Object (RD)	-	1-32767	-
Security-Label	Object (Security Label)	-	0-1	-

[1]As defined in the **XDS Specification** (see **Referenced Documents**).

Table 5-7 OM Attributes of Auto-Forward-Arguments

Alternate Recipient Allowed

Whether the originator permits the MTS to deliver the subject message to an alternate recipient. An MD may (but need not) assign a user, the alternate recipient, to accept delivery of messages whose Recipient Descriptors attributes contain O/R names that are invalid but recognised as meant to denote users of that MD.

Confidentiality Algorithm

Identifies the algorithm that the originator of the submitted message used to encrypt its content and which the recipients may use to decrypt it.

The algorithm may be either symmetric or asymmetric. If the former, the associated key may be derived from the Token attribute of any of the submitted message's RDs or, alternatively, distributed by some other means. If the latter, the originator may use the intended recipient's public key to encrypt the content, and the recipient may use the associated secret key to decrypt it. The submitted message must be addressed to either a single recipient or a group of recipients sharing the same key pair.

Content Correlator

Information facilitating the correlation with the submitted communique of any reports it may provoke. This attribute is present at the option of the originator's UA. It is not conveyed to recipients at delivery.

Content Identifier
Information facilitating the correlation with the submitted communique of any reports it may provoke. This attribute is present at the option of the originator's UA. It is conveyed to recipients at delivery.

Content Return Requested
Whether the Content attribute is to be included as the like-named attribute of any NDRs the submitted message provokes.

Conversion Loss Prohibited
Whether the originator prohibits the MTS from converting the subject message (should such conversion be necessary) if it would cause loss of information as defined in X.408.

Conversion Prohibited
Whether the originator prohibits the MTS from converting the subject message (should such conversion be necessary) under any circumstances.

Deferred Delivery Time
The date and time, if any, before which the submitted message shall not be delivered. Delivery deferral is normally the responsibility of the MD that originates the submitted message. Thus messages whose Deferred Delivery Time attributes are present shall be transferred between MDs only by bilateral agreement between those MDs.

Disclosure Allowed
Whether the O/R names of other recipients are to be indicated to each recipient at delivery.

Expansion Prohibited
Whether the originator instructs the MTS to issue an NDR rather than expand a DL if the O/R name specified for any of the recipients proves to denote a DL not a user.

Latest Delivery Time
The date and time after which the MTS is to treat the submitted message as undeliverable if it has not yet been delivered to a particular recipient.

Origin Check
A means by which a third party (e.g., a user or an MTA) can verify the submitted communique's origin. This attribute is present at the option of the originator's UA. The algorithm involved is applied to an instance of the Origin Check Basis class.

Original EITs
The EITs of the Content attribute of the subject message. This attribute is present at the option of the originator's UA.

Originator Certificate
The originator's certificate. Generated by a trusted source (for example, a CA), it constitutes a verified copy of the originator's PAEK. This attribute is present at the option of the originator's UA.

Originator Name
The O/R name of the submitted communique's originator.

Originator Return Address
The postal O/R address of the submitted message's originator. It shall be present if the originator supplied a postal O/R address for an intended recipient or included physical delivery among a recipient's preferred delivery modes. It may also be present if a recipient DL contains, or is likely to contain, one or more members for whom physical delivery is required.

Priority The relative priority at which the submitted message is to be transferred. For its defined values, see **Section 5.3.9**, **Priority**.

Proof of Submission Requested
Whether the originator of the submitted message requires proof of its submission.

Reassignment Prohibited
Whether the originator prohibits the intended recipients from redirecting the submitted communique.

Recipient Descriptors
The RDs of the submitted communique's intended recipients.

Security Label
The security label associated with the submitted communique. It must be assigned in line with the security policy in force.

5.14 Auto-Forward-Registration-Parameter

An instance of OM class **Auto-Forward-Registration-Parameter** specifies the criteria to determine whether a delivered message should be forwarded.

An instance of the OM class **Auto-Forward-Registration-Parameter** has the OM attributes of its superclasses: *Object*, and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Auto-Forward-Arguments	Object(Auto-Forward-Arguments)	-	1	-
Delete-After-Auto-Forward	Boolean	-	1	true
Filter	Object(Filter)	-	0-1	-
Other-Parameters	String(Octet)	-	0-1	-

Table 5-8 OM Attributes of Auto-Forward-Registration-Parameter

Auto-Forward-Arguments
The set of arguments to be used for each auto-forward message-submission operation.

Delete-After-Auto-Forward
If true, an entry is to be deleted after auto-forwarding. If false, it is not deleted.

Filter
The set of criteria which a delivered message must satisfy for the message to be auto-forwarded with the given set of parameters.

Other-Parameters
Optional extra information to be used in auto-forwarding.

5.15 Bind-Argument

An instance of OM class **Bind-Argument** specifies information necessary for establishing a session with the Message Store, together with details of the service required.

An instance of OM class **Bind-Argument** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Initiator	Object(OR-Name) †	-	1	-
Initiator-Credentials	Object(Credentials)	-	1	-
Security-Context	Object(Security-Label)	-	0/more	-
Fetch-Restrictions	Object(Restrictions)	-	0-1	-
MS-Configuration-Request	Boolean	-	1	false

† As defined in the X.400 Specification (see reference **X.400**).

Table 5-9 OM Attributes of Bind-Argument

Initiator
This specifies the OR-Name of the initiator (i.e., the UA) of this session (or association) with the MS.

Initiator-Credentials
This specifies the credentials of the initiator for authentication purposes.

Security-Context
This identifies the security context at which the initiator proposes to operate.

Fetch-Restrictions
This specifies the restrictions on entries to be returned as result of a **Fetch**( ) function. These restrictions prevail until the **Unbind**( ) function is issued.

MS-Configuration-Request
If true, this specifies the request to obtain information relating to which auto-actions and optional attributes the MS provides support for. If false, no such request is being made.

5.16 Bind-Result

An instance of OM class **Bind-Result** describes the result returned from the **Bind**( ) function. It provides information about the MS capabilities, if requested.

An instance of OM class **Bind-Result** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Responder-Credentials	Object(Credentials)	-	1	-
Available-Auto-Actions	String(Object-Identifier)	-	0/more	-
Available-Attribute-Types	String(Object-Identifier)	-	0/more	-
Alert-Indication	Boolean	-	1	false
Content-Types-Supported	String(Object-Identifier)	-	0/more	-

Table 5-10 OM Attributes of Bind-Result

Responder-Credentials
This contains the credentials of the MS.

Available-Auto-Actions
This specifies the set of all possible auto-actions that are supported by the MS (not just those requested by the UA). This is only present if the MS-Configuration-Request was made for the **Bind**( ) function.

Available-Attribute-Types
This specifies the set of all optional MS attribute-types that are supported by the MS. This is only present if the MS-Configuration-Request was made for the **Bind**( ) function.

Alert-Indication
If true, this indicates an alert condition has occurred since the last successful Alert-indication.

Content-Types-Supported
This specifies a set of object-identifiers defining the content-types of which the MS has knowledge. This is only present if the MS-Configuration-Request was made for the **Bind**( ) function.

5.17 Change-Credentials

An instance of OM class **Change-Credentials** gives the user's current (old) credentials and the new credentials to which the user would like to change.

An instance of OM class **Change-Credentials** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Old-Credentials	Object(Credentials)	-	1	-
New-Credentials	Object(Credentials)	-	1	-

Table 5-11 OM Attributes of Change-Credentials

Old-Credentials
This specifies the user's current credentials. This may be credentials for either simple or strong authentication.

New-Credentials
This specifies the credentials to which the user would like to change.

Note: This shall be the same type (i.e., simple or strong) as the **Old-Credentials**.

5.18 Check-Alert-Result

An instance of OM class **Check-Alert-Result** gives information regarding an alert when a new entry has been entered into the Message Store in response to *Check-Alert*( ).

An instance of OM class **Check-Alert-Result** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Alert-Registration-Identifier	Integer	-	1	-
New-Entry	Object(MS-Entry-Information)	-	0-1	-

Table 5-12 OM Attributes of Check-Alert-Result

Alert-Registration-Identifier
This identifies which auto alert registrations resulted in the alert.

New-Entry
If present, this conveys the information from the new entry which was requested in the auto alert registration parameter. This is absent if the user did not specify an auto alert registration parameter.

5.19 Common-Controls

The OM class **Common-Controls** indicates the control parameters, which are related to delivery and submission, common to the classes Default-Delivery-Controls, Delivery-Controls and Submission-Controls (although the latter two are not used by functions defined in this specification).

The OM class **Common-Controls** is an abstract class and has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Restrict	Boolean	-	1	true
Permissible-Operations	String(Bit)	-	0-1	-
Permissible-Maximum-Content-Length	Integer	-	0-1	-
Permissible-Lowest-Priority	Enum(Priority) †	-	0-1	-

† As defined in the X.400 Specification (see reference **X.400**).

Table 5-13 OM Attributes of Common-Controls

Note: The following control parameters apply either to delivery or to submission (the latter being indicated in brackets [ ]). For further details, see sections 8.3.1.3.1 and 8.2.1.4.1 in reference **X.411**.

Restrict

This indicates whether the controls on the delivery-port [or submission-port] abstract operations are to be updated (‘true’) or removed (‘false’).

Permissible-Operations

This indicates the abstract operations that the MTS may invoke on the MS or UA.

Permissible-Maximum-Content-Length

This indicates the content-length, in octets, of the longest content message that the MTS shall deliver to the MS or UA via the Message-Delivery abstract operation [or the MS or UA shall submit to the MTS via the Message-Submission abstract operation].

Permissible-Lowest-Priority

This indicates the priority of the lowest priority message that the MTS shall deliver to the MS or UA via the Message-Delivery abstract operation [or the MS or UA shall submit to the MTS via the Message-Submission abstract operation].

5.20 Creation-Time-Range

An instance of OM class **Creation-Time-Range** identifies a contiguous sequence of entries based on their times of creation.

An instance of OM class **Creation-Time-Range** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
From	String(UTC-Time)	-	0-1	-
To	String(UTC-Time)	-	0-1	-

Table 5-14 OM Attributes of Creation-Time-Range

From

This indicates creation-time that is the lower bound for the range. If absent, the default is no lower bound; and the selection begins with the entry with the earliest creation-time in the information base.

To

This indicates creation-time that is the upper bound for the range. If absent, the default is no upper bound; and the selection finishes with the entry with the latest creation-time in the information base.

5.21 Credentials

An instance of OM class **Credentials** holds either a simple password when simple authentication is used or strong credentials when strong authentication is used.

An instance of OM class **Credentials** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Simple	Object(Password)	-	0-1 †	-
Strong	Object(Strong-Credentials)	-	0-1 †	-

† No instance will contain both of the above OM attributes.

Table 5-15 OM Attributes of Credentials

Simple

This is the password used for simple authentication.

Strong

This specifies the credentials used for strong authentication.

5.22 Default-Delivery-Controls

An instance of OM class **Default-Delivery-Controls** gives the default control parameters related to delivery.

An instance of OM class **Default-Delivery-Controls** has the OM attributes of its superclasses: *Object*, **Common-Controls** and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Permissible-Content-Types-Int	Integer †	-	0/more	-
Permissible-Content-Types	String(Object-Identifier) †	-	0/more	-
Permissible-EITs	Object(EITs) ‡	-	0-1	-

† For services based on MHS-1984 (see reference **MHS-1984**), only attribute "Permissible-Content-Types-Int" is available. For services based on MHS-1988 (see reference **MHS-1988**), either "Permissible-Content-Types-Int" or "Permissible-Content-Types" is available.

‡ As defined in the X.400 Specification (see reference **X.400**).

Table 5-16 OM Attributes of Default-Delivery-Controls

Permissible-Content-Types-Int
Permissible-Content-Types

This indicates the content-types that shall appear in messages the MTS shall deliver to the MS via the Message-Delivery abstract operation. **Content-Type** identifies the syntax and semantics of the value of the OM attribute Content of the OM class Message. Its defined values are as prescribed for the like-named attribute specific to the OM class Communique of the Message Handling Package of the X.400 Specification (see reference **X.400**).

Permissible-EITs

This indicates the encoded-information-types that shall appear in messages the MTS shall deliver to the MS via the Message-Delivery abstract operation.

5.23 Delete-Argument

An instance of OM class **Delete-Argument** describes the arguments for the **Delete**( ) function.

An instance of OM class **Delete-Argument** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Information-Base-Type	Integer	-	0-1	stored-messages
Items	Object(Items)	-	1	-

Table 5-17 OM Attributes of Delete-Argument

Information-Base-Type

This specifies which information base type is being addressed (see Section 6.3.1 in reference **X.413**). Its value must be one of the following:

— **stored-messages** *{MS_STORED_MESSAGES}*
This type specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**).

— **inlog** *{MS_INLOG}*

— **outlog** *{MS_OUTLOG}*

The default is **stored-messages** *{MS_STORED_MESSAGES}*.

Items

This determines which entries are to be deleted.

5.24 Deliverable-Content-Types

An instance of OM class **Deliverable-Content-Types** indicates which content-types that the MTS shall permit to appear in messages delivered to the MS.

An instance of OM class **Deliverable-Content-Types** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Content-Type-Int	Integer †	-	1/more	-
Content-Type	String(Object-Identifier) †	-	1/more	-

† For services based on MHS-1984 (see reference **MHS-1984**), only attribute Content-Type-Int" is available. For services based on MHS-1988 (see reference **MHS-1988**), either "Content-Type-Int" or "Content-Type" is available.

Table 5-18 OM Attributes of Deliverable-Content-Types

Content-Type-Int
Content-Type

This identifies the syntax and semantics of the value of the OM attribute Content of the OM class Message. Its defined values are as prescribed for the like-named attribute specific to the OM class Communique of the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.25 EITs

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.26 Fetch-Argument

An instance of OM class **Fetch-Argument** describes the arguments for the **Fetch**( ) function.

An instance of OM class **Fetch-Argument** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Information-Base-Type	Integer	-	0-1	stored-messages
Item	Object(Item)	-	1	-
Requested-Attributes	Object(MS-Entry-Information-Selection)	-	0-1	-

Table 5-19 OM Attributes of Fetch-Argument

Information-Base-Type

This specifies which information base type is being addressed (see Section 6.3.1 in reference **X.413**). Its value must be one of the following:

— **stored-messages** *{MS_STORED_MESSAGES}*
This type specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**.)

— **inlog** *{MS_INLOG}*

— **outlog** *{MS_OUTLOG}*

The default is **stored-messages** *{MS_STORED_MESSAGES}*.

Item

This determines which entry is to be fetched.

Requested-Attributes

This indicates what information from the selected entry is to be returned in the result.

Note: If absent, it implies that information about the entry itself, rather than the attributes of the entry, is requested.

5.27 Fetch-Attribute-Defaults

An instance of OM class **Fetch-Attribute-Defaults**, for the **Register-MS**( ) function, specifies a default set of attribute-types to indicate which attributes should be returned for any subsequent **Fetch**( ) functions if the Entry-Information-Selection argument is absent. This value replaces any previously registered default set.

An instance of OM class **Fetch-Attribute-Defaults** has the OM attributes of its superclasses: *Object*, **Attribute-Defaults**, and no additional OM attributes.

5.28 Fetch-Result

An instance of OM class **Fetch-Result** gives the result of a successful **Fetch**( ) function.

An instance of OM class **Fetch-Result** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Entry-Information	Object(MS-Entry-Information)	-	0-1	-
List	Integer	-	0/more	-
Next	Integer	-	0-1	-

Table 5-20 OM Attributes of Fetch-Result

Entry-Information

This is the set of MS attributes from one entry as requested in the argument to the **Fetch**( ) function. This is not present if a search was performed but no entry was selected.

List

This is returned in the case that a search was performed and more than one entry was found that matched the search selector. This would then give the sequence-numbers, in ascending order, of these further entries.

Next

This is returned in the case where the number of entries selected would have been greater if it were not for the limit specified in the selector. This would then give the sequence-number for the next entry that would have been selected.

5.29 Filter

As defined in the Directory Services Package in the XDS Specification (see reference **XDS**).

Note: The following points out some differences regarding the interpretation of filters as defined by the Directory and the MS standards. Consequently, these are additional constraints on the OM class **Filter** in the context of MS.

a. In the MS, a Filter-Item may only be **true** or **false**, whereas it may be **true**, **false** or undefined in the Directory.

b. In the MS, the definitions of the **greater-or-equal** and **less-or-equal** Filter-Item-Types are the opposite of those in the Directory. Furthermore, they fail to yield **true** in the event of equality.

For further details, refer to the CCITT Special Rapporteur Q18/VII MHS Implementor's Guide (see reference **MHS**).

5.30 Filter-Item

As defined in the Directory Services Package in the XDS Specification (see reference **XDS**).

See also the note under Section 5.29 on page 83.

5.31 Item

An instance of the OM class **Item** identifies the criterion for selecting a single MS entry to be returned by the **Fetch**( ) function.

An instance of the OM class **Item** has the OM attributes of its superclasses: *Object*, **Items** and no additional OM attributes.

Additional Constraints

1. **Selector**
   If present, this specifies a set of entries of which the one with the lowest sequence-number is the entry to be chosen.
2. **Precise**
   If present, there shall be precisely a single value which would identify an MS entry by its sequence-number.

5.32 Item-To-Forward

An instance of the OM class **Item-To-Forward** gives the sequence-number identifying the single entry that is to be forwarded by the **MS-Submit**( ) function.

Note: The entry to be forwarded should be a delivered message entry. Forwarding of entries that are not delivered messages is not defined in this specification.

An instance of the OM class **Item-To-Forward** has the OM attributes of its superclasses: *Object*, **Submitted-Message** and the additional OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Sequence-Number	Integer	-	1	-

Table 5-21 OM Attributes of Item-To-Forward

Sequence-Number
This identifies the MS entry, by its sequence number, that is to be forwarded via indirect submission.

5.33 Items

An instance of the OM class **Items** identifies the criterion for selecting MS entries to be removed by the **Delete**( ) function.

An instance of the OM class **Items** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Selector	Object(Selector)	-	0-1 †	-
Precise	Integer	-	0/more †	-

† No instance will contain both of the above OM attributes.

Table 5-22 OM Attributes of Items

Selector
If present, this specifies a set of entries to be chosen.

Precise
If present, this is a list of sequence-number(s) to precisely identify the entries to be chosen based on their sequence-numbers.

5.34 Label-And-Redirection

An instance of OM class **Label-And-Redirection** indicates a set of security-label and the OR-Name of an alternate-recipient.

An instance of OM class **Label-And-Redirection** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Security-Label	Object(Security-Label) †	-	0-1	-
Alternate-Recipient	Object(OR-Name) †	-	0-1	-

† As defined in the X.400 Specification (see reference **X.400**).

Table 5-23 OM Attributes of Label-And-Redirection

Security-Label
This identifies the user-security-label.

Alternate-Recipient
This identifies the OR-Name of an alternate-recipient.

5.35 Labels-And-Redirections

An instance of OM class **Labels-And-Redirections** indicates a set of security-label and the OR-Name of alternate-recipient pair.

An instance of OM class **Labels-And-Redirections** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Label-And-Redirection	Object(Label-And-Redirection)	-	1/more	-

Table 5-24 OM Attributes of Labels-And-Redirections

Label-And-Redirection

This identifies either one or both of the following:

— a security-label

— the OR-Name of an alternate-recipient

5.36 List-Argument

An instance of OM class **List-Argument** describes the arguments for the **List**( ) function.

An instance of OM class **List-Argument** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Information-Base-Type	Integer	-	0-1	stored-messages
Selector	Object(Selector)	-	1	-
Requested-Attributes	Object(MS-Entry-Information-Selection)	-	0-1	-

Table 5-25 OM Attributes of List-Argument

Information-Base-Type

This specifies which information base type is being addressed (see Section 6.3.1 in reference **X.413**). Its value must be one of the following:

— **stored-messages** *{MS_STORED_MESSAGES}*
This type specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**.)

— **inlog** *{MS_INLOG}*

— **outlog** *{MS_OUTLOG}*

The default is **stored-messages** *{MS_STORED_MESSAGES}*.

Selector

This determines which entries are to be listed.

Requested-Attributes

This indicates what information from the selected entry is to be returned in the result.

Note: If absent, it implies that information about the entry itself, rather than the attributes of the entry, is requested.

5.37 List-Attribute-Defaults

An instance of OM class **List-Attribute-Defaults**, for the **Register-MS**( ) function, specifies a default set of attribute-types to indicate which attributes should be returned for any subsequent **List**( ) function if the Entry-Information-Selection argument is absent. This value replaces any previously registered default set.

An instance of OM class **List-Attribute-Defaults** has the OM attributes of its superclasses: *Object*, **Attribute-Defaults**, and no additional OM attributes.

5.38 List-Result

An instance of OM class **List-Result** gives the result of a successful **List**( ) function.

An instance of OM class **List-Result** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Next	Integer	-	0-1	-
Requested	Object(MS-Entry-Information)	-	0/more	-

Table 5-26 OM Attributes of List-Result

Next
This is returned in the case where the number of entries selected would have been greater if it were not for the limit specified in the selector. This would then give the sequence-number for the next entry that would have been selected.

Requested
This is the set of MS attributes of the entries requested in the argument to the **List**( ) function. This is not present if a search was performed but no entries were selected.

5.39 MS-Entry-Information

An instance of OM class **MS-Entry-Information** describes information about an MS entry.

An instance of OM class **MS-Entry-Information** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Sequence-Number	Integer	-	1	-
Attributes	Object(Attribute)	-	0/more	-

Table 5-27 OM Attributes of MS-Entry-Information

Sequence-Number
This identifies precisely the entry that is to be chosen.

Attributes
This identifies the MS attributes, or those requested, of an MS entry.

5.40 MS-Entry-Information-Selection

An instance of OM class **MS-Entry-Information-Selection** indicates which MS attributes of an entry are being requested.

An instance of OM class **MS-Entry-Information-Selection** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Selection	Object(Attribute-Selection)	-	1/more	-

Table 5-28 OM Attributes of MS-Entry-Information-Selection

Selection
This indicates which MS attributes of an entry are being requested.

5.41 MTS-Identifier

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.42 OR-Name

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.43 Password

An instance of OM class **Password** gives the simple password for simple authentication.

An instance of OM class **Password** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
IA5-String	String(IA5)	-	0-1 †	-
Octet-String	String(Octet)	-	0-1 †	-

† No instance will contain more than one of the above OM attributes.

Table 5-29 OM Attributes of Password

IA5-String
This specifies the simple password in IA5 string format.

Octet-String
This specifies the simple password in Octet string format.

5.44 Range

An instance of OM class **Range** is used to select a contiguous sequence of entries, based on either their sequence-numbers or creation-times, from an information base (see Section 8.1.1 in reference **X.413**.)

An instance of OM class **Range** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Sequence-Number-Range	Object(Sequence-Number-Range)	-	0-1 †	-
Creation-Time-Range	Object(Creation-Time-Range)	-	0-1 †	-

† No instance will contain both of the above OM attributes.

Table 5-30 OM Attributes of Range

Sequence-Number-Range
This identifies a contiguous sequence of entries based on their sequence-numbers.

Creation-Time-Range
This identifies a contiguous sequence of entries based on the times the entries were created.

5.45 Register-Argument

An instance of OM class **Register-Argument** specifies how to modify (via the **Register**( ) function) various parameters held by the MTS regarding delivery of messages to the MS.

An instance of OM class **Register-Argument** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Name	Object(OR-Name) †	-	0-1	-
Address	Object(Address) ‡	-	0-1	-
Deliverable-EIT	Object(EITs) †	-	0-1	-
Deliverable-Maximum-Content-Length	Integer	-	0-1	-
Default-Delivery-Controls	Object(Default-Delivery-Controls)	-	0-1	-
Deliverable-Content-Types	Object(Deliverable-Content-Types)	-	0-1	-
Labels-And-Redirections	Object(Labels-And-Redirections)	-	0-1	-

† As defined in the X.400 Specification (see reference **X.400**).

‡ As defined in the XDS Specification (see reference **XDS**).

Table 5-31 OM Attributes of Register-Argument

Name

This specifies the OR-Name of the MS (this corresponds to the OR-Name of the UA), if the user-name is to be changed.

Address

This specifies the address of the MS, if it is required by the MTS and if it is to be changed. This may contain one of the following forms of address:

— the X.121-address and/or the TSAP-ID (transport service access point identifier).

— the PSAP-ID (presentation service access point identifier).

Deliverable-Encoded-Information-Types

This indicates the encoded-information-types that the MTS shall permit to appear in messages delivered to the MS, if they are to be changed. The MTS shall reject as undeliverable any message for an MS for which the MS is not registered to accept delivery of all of the encoded-information-types of the message. Note that the MS may register to receive the **undefined** encoded-information-type. This OM attribute, **Deliverable-Encoded-Information-Types** also indicates the possible encoded-information-types to which implicit conversion can be performed.

Deliverable-Maximum-Content-Length

This indicates the content-length, in octets, of the longest content message that the MTS shall permit to appear in messages being delivered to the MS, if it is to be changed. The MTS shall reject as undeliverable any message for an MS for which the MS is not registered to accept delivery of messages of its size.

Default-Delivery-Controls

This indicates default delivery controls which are registered using the **Register**( ) function. The default delivery control OM attributes shall not admit messages whose delivery are prohibited by the prevailing registered values of the **Deliverable-Encoded-Information-Types** OM attribute, the **Deliverable-Content-Types** OM attribute or the **Deliverable-Maximum-Content-Length** OM attribute.

Deliverable-Content-Types

This indicates the content-types that the MTS shall permit to appear in messages delivered to the MS, if they are to be changed. The MTS shall reject as undeliverable any message for an MS for which the MS is not registered to accept delivery of all of the content-types of the message. Note that the MS may register to receive the undefined content-type.

Labels-And-Redirections

This contains either one or both of the following:

— the OR-Name of an alternate recipient to which messages are to be redirected, if this is to be changed.

— the security-label of the UA, if they are to be changed.

5.46 Register-MS-Argument

An instance of OM class **Register-MS-Argument** specifies the information to be registered or de-registered through the **Register-MS**( ) function.

An instance of OM class **Register-MS-Argument** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Auto-Action-Registrations	Object(Auto-Action-Registration)	-	0/more	-
Auto-Action-Deregistrations	Object(Auto-Action-Deregistration)	-	0/more	-
List-Attribute-Defaults	Object(List-Attribute-Defaults)	-	0-1	-
Fetch-Attribute-Defaults	Object(Fetch-Attribute-Defaults)	-	0-1	-
Change-Credentials	Object(Change-Credentials)	-	0-1	-
User-Security-Labels	Object(Security-Label)	-	0/more	-

Table 5-32 OM Attributes of Register-MS-Argument

Auto-Action-Registrations
This is a set of auto-action-registrations, one for each auto-action to be registered.

Auto-Action-Deregistrations
This is a set of auto-action-deregistrations, one for each auto-action to be deregistered.

List-Attribute-Defaults
This specifies a default set of attribute types to indicate which attributes should be returned for any subsequent **List**( ) function if the Requested-Attributes OM attribute of List-Argument is absent.

Fetch-Attribute-Defaults
This specifies a default set of attribute types to indicate which attributes should be returned for any subsequent **Fetch**( ) function if the Requested-Attributes OM attribute of Fetch-Argument is absent.

Change-Credentials
If change credentials is requested, this specifies the old and new credentials of the end user.

User-Security-Labels
This contains the security-labels of the UA, if they are to be changed.

5.47 Restrictions

An instance of OM class **Restrictions** describes the restrictions on entries to be returned as result of a **Fetch**( ) function.

An instance of OM class **Restrictions** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Allowed-Content-Types	String(Object-Identifier)	-	0/more	-
Allowed-EITs	String(Object-Identifier)	-	0/more	-
Max-Content-Length	Integer	-	0-1	-

Table 5-33 OM Attributes of Restrictions

Allowed-Content-Types

This specifies the content-types that the UA is prepared to accept as a result of a **Fetch**( ) function invocation. Any message with a content-type other than the ones specified will not be returned; but result in an error, unless the **Fetch**( ) function has explicitly overridden the restriction. If absent, the default is that no fetch-restrictions on content-types need to be performed.

Allowed-EITs

This specifies the encoded-information-types that the UA is prepared to accept as a result of a **Fetch**( ) function. For any message with an EIT other than the ones specified, a filtering will occur so that disallowed EIT parts are not returned along with the text of the message. If the entire message consists of disallowed EITs, an error will be reported. No filtering will occur if the **Fetch**( ) function has been explicitly overridden. If absent, the default is that no fetch-restrictions on EITs need to be performed.

Max-Content-Length

This specifies the maximum content length that the UA is prepared to accept as a result of a **Fetch**( ) function invocation. Any message with a content-length exceeding this maximum will not be returned but will result in an error, unless the **Fetch**( ) function has explicitly overridden the restriction. If absent, the default is that no fetch-restrictions on content-length need to be performed.

5.48 Security-Label

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.49 Selector

An instance of OM class **Selector** is used to describe the criteria for selecting entries from an information base where the selection operates in three stages, as follows:

- Firstly, the total set of entries in the information base may be restricted to a particular contiguous set by specifying its range.
- Secondly, entries from within this set may be selected by specifying a filter which the selected entry must satisfy.
- Thirdly, a limit may be imposed on the number of entries thus selected; in this case, it is those entries with the lowest sequence-numbers which are selected.

An instance of OM class **Selector** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Child-Entries	Boolean	-	1	false
Range	Object(Range)	-	0-1	-
Filter	Object(Filter)	-	0-1	-
Limit	Integer	-	0-1	-
Override	String(Bit)	-	0-1	-

Table 5-34 OM Attributes of Selector

Child-Entries

This indicates whether or not child entries are considered for selection. If **false**, only the main-entries are considered for selection. If **true**, both main-entries and child-entries are considered for selection.

Range

This specifies the contiguous set of entries selected in the first stage of the selection process as mentioned above. If absent, the default is *unbounded*.

Filter

This specifies the filter (or selection criterion) for the second stage of the selection process as mentioned above. If absent, the default is *all entries within the specified range*.

Limit

This is the upper limit on the number of entries selected in the third stage of the selection process as mentioned above. If absent, the default is all entries are returned.

Override

See Section 8.1.3 in reference **X.413**. This identifies any override restrictions, if required.

5.50 Sequence-Number-Range

An instance of OM class **Sequence-Number-Range** identifies a contiguous sequence of entries based their sequence-numbers.

An instance of OM class **Sequence-Number-Range** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
From-Int	Integer	-	0-1	-
To-Int	Integer	-	0-1	-

Table 5-35 OM Attributes of Sequence-Number-Range

From-Int

This indicates the sequence-number that is the lower bound for the range. If absent, the default is *no lower bound*; and the selection begins with the entry with the lowest sequence-number in the information base.

To-Int

This indicates the sequence-number that is the upper bound for the range. If absent, the default is *no upper bound*; and the selection finishes with the entry with the highest sequence-number in the information base.

5.51 Session

An instance of OM class **Session** provides information for a particular association between a client program and the Message Store.

An instance of OM class **Session** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Initiator	Object(OR-Name) †	-	1	-
File-Descriptor	Integer	-	0-1	(see below)

† As defined in the X.400 Specification (see reference **X.400**).

Table 5-36 OM Attributes of Session

Initiator

This specifies the OR-Name of the initiator (i.e., the UA) of this session (or association) with the MS.

File-Descriptor (Optional Functionality)

This indicates the file descriptor associated with the session. The file descriptor may be used in subsequent calls to vendor-specific system facilities to suspend the process (e.g., UNIX System V **poll**( ) or Berkeley Source Distribution **select**( )). Its use for other purposes is unspecified.

5.52 Strong-Credentials

An instance of OM class **Strong-Credentials** gives the credentials for strong authentication.

An instance of OM class **Strong-Credentials** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Bind-Token	Object(Asymmetric-Token) †	-	0-1	-
Certificate	Object(Certificates) ‡	-	0-1	-

† As defined in the X.400 Specification (see reference **X.400**).

‡ As defined in the XDS Specification (see reference **XDS**).

Table 5-37 OM Attributes of Strong-Credentials

Bind-Token
This specifies the token used to convey to the protected security-relevant information.

Certificate
This specifies the certificate used to convey a verified copy of the public asymmetric encryption key of the subject of the certificate.

5.53 Submission-Results

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.54 Submitted-Communique

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.55 Submitted-Message

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.56 Submitted-Probe

As defined in the Message Handling Package in the X.400 Specification (see reference **X.400**).

5.57 Summarize-Argument

An instance of OM class **Summarize-Argument** specifies the argument for the **Summarize**( ) function.

An instance of OM class **Summarize-Argument** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Information-Base-Type	Integer	-	0-1	stored-messages
Selector	Object(Selector)	-	1	-
Summary-Requests	Object(Summary-Requests)	-	0-1	-

Table 5-38 OM Attributes of Summarize-Argument

Information-Base-Type

This specifies which information base type is being addressed (see Section 6.3.1 in reference **X.413**). Its value must be one of the following:

stored-messages *{MS_STORED_MESSAGES}*
This type specifies the repository containing entries for delivered messages and reports (see Section 6.4 in reference **X.413**.)

— **inlog** *{MS_INLOG}*

— **outlog** *{MS_OUTLOG}*

The default is **stored-messages** *{MS_STORED_MESSAGES}*.

Selector

This is the set of criteria for determining which entries shall be summarised.

Summary-Requests

This indicates the sequence of attribute-types for which summaries are requested.

5.58 Summary

An instance of OM class **Summary** describes part of the result returned by the **Summary**( ) function; it summarises the count of entries based on the presence (and then, based on the actual attribute-values) or absence of certain various attribute-types of the entries, as requested.

An instance of OM class **Summary** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Absent	Integer	-	0-1	-
Present	Object(Summary-Present)	-	0/more	-

Table 5-39 OM Attributes of Summary

Absent

This is a count of the entries that do not contain an attribute of the attribute-type specified in the request. This is absent if there are no such entries.

Present

This gives a list of summaries for the entries that do contain an attribute of the attribute-type specified, broken down by the attribute-values actually present. This is absent if there are no such entries.

5.59 Summary-Present

An instance of OM class **Summary-Present** describes part of the result returned by the **Summary**( ) function; in particular, it gives the count of entries that contain attributes of the specified attribute-type for the specified attribute-value.

An instance of OM class **Summary-Present** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
AVA	Object(AVA)	-	1	-
Count	Integer	-	0-1	-

Table 5-40 OM Attributes of Summary-Present

AVA

This specifies an attribute value assertion (i.e., it specifies the attribute-type and a particular attribute-value); the **Summary**( ) function returns, in **Count**, the total of the entries satisfying this attribute value assertion.

Count

This is a count of the entries which do contain the attribute of the specified attribute-type and whose value matches the specified attribute-value. This is absent if there are no such entries.

5.60 Summary-Requests

An instance of OM class **Summary-Requests** describes the sequence of attribute-types for which summaries are requested in the **Summary**( ) function.

An instance of OM class **Summary-Requests** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Requested-Str	String(Object-Identifier)	-	0/more	-

Table 5-41 OM Attributes of Summary-Requests

Requested-Str
This is the sequence of attribute-types for which summaries are requested. This is only present if a summary is requested.

5.61 Summary-Result

An instance of OM class **Summary-Result** gives the result of a successful **Summary**( ) function.

An instance of OM class **Summary-Result** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Next	Integer	-	0-1	-
Count	Integer	-	1	-
Span	Object(Sequence-Number-Range)	-	0-1	-
Summaries	Object(Summary)	-	0/more	-

Table 5-42 OM Attributes of Summary-Result

Next
This is returned in the case where the number of entries selected would have been greater if it were not for the limit specified in the selector. This would then give the sequence-number for the next entry that would have been selected.

Count
This gives the number of entries that matched the selection criteria.

Span
This gives the range of sequence-numbers of entries that matched the selection criteria. It is absent if there were no such entries (i.e., **Count** value is zero).

Summaries
This is a sequential list of summaries; one for each summary-request. The summaries are returned in the order that they were requested in the **Summary**( ) function.

5.62 Wait-Result

An instance of OM class **Wait-Result** gives the result of a successful **Wait**( ) function.

An instance of OM class **Wait-Result** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Wait-New-Available	Boolean	-	1	true

Table 5-43 OM Attributes of Wait-Result

Wait-New-Available

This indicates whether any new MS entries have become available for retrieval. Its value is true if new entries in the MS have become available for retrieval, and false if otherwise.

Chapter 6

Errors

6.1 Introduction

This chapter defines the errors that can arise in the use of the interface and describes the method used to report them.

Errors are reported to the application program by means of the **Status** that is returned by most functions (it is the function result in the C language binding). A function which completes successfully returns the value Success *{MS_SUCCESS}*. When a function is not successful it will return a private object that contains details about the problem which prevented its successful completion, unless no workspace exists (that is, if **Initialize**( ) has not been called successfully) in which case the function returns the error constant **no-workspace***[MS_NO_WORKSPACE]*.

There are two types of failure for asynchronous operations. The first type is reported immediately in the status of the invoking function whereas the second type is returned as the **Operation-Status** result of a later call to **Receive-Result**( ). However, the distinction between these two types of failure is implementation-dependent.

Errors are classified into several OM classes. The Standards (see reference **X.413**) classify errors into several different kinds.

The interface also defines three more kinds of error:

- Library-Error
- Communications-Error
- System-Error

Each of these is represented by an OM class and these are detailed below in alphabetical order. All of them inherit the OM attribute **Problem** from their superclass *Error*, which is described first.

All the OM classes defined in this chapter are part of the Message Store package introduced in the previous chapter. The possible errors that each interface function may generate are listed for the respective interface functions in Chapter 4. Errors are mandatory only if the applicable FU is supported.

6.2 OM Class Hierarchy

This section depicts the hierarchical organisation of the classes defined in this chapter and thus, indicates the inheritance of additional OM attributes from its superclass(es) by each class. Subclassification is indicated by indentation and the names of abstract classes are rendered in italics. Thus, for instance, the concrete class **Attribute-Problem** is an immediate subclass of the abstract class *Error* which in turn is an immediate subclass of the abstract class *Object*.

Object (defined in the XOM Specification - see reference **XOM**)

- Attribute-Error
- Auto-Action-Request-Error
- Delete-Error
- Fetch-Restriction-Error
- Sequence-Number-Error
- *Error*
  - Attribute-Problem
  - Auto-Action-Request-Problem
  - Bind-Error
  - Cancel-Submission-Error
  - Communications-Error
  - Delete-Problem
  - Element-Of-Service-Not-Subscribed-Error
  - Fetch-Restriction-Problem
  - Inconsistent-Request-Error
  - Invalid-Parameters-Error
  - Library-Error
  - Originator-Invalid-Error
  - Range-Error
  - Recipient-Improperly-Specified-Error
  - Register-Rejected-Error
  - Remote-Bind-Error
  - Security-Error
  - Service-Error
  - Sequence-Number-Problem
  - Submission-Control-Violated-Error
  - System-Error
  - Unsupported-Critical-Function-Error

The client program is not permitted to create or modify any instances of these OM classes. Furthermore, this specification does not mandate that any OM classes be translatable using **OM_Encode**( ) and **OM_Decode**( ).

Note that an **Attribute-Error**, not a subclass of *Error*, is special in that it may report several problems at a time; each problem is reported in an **Attribute-Problem**, which is a subclass of *Error*. The situation is similar for **Auto-Action-Request-Error**, **Delete-Error** and **Fetch-Restriction-Error** along with their corresponding **Problem** classes.

6.3 Error

The OM class **Error** consists of parameters common to all errors.

It is an abstract class which has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Problem	Enum(Problem)	-	1	-

Table 6-1 OM Attributes of Error

Problem

This indicates the type of error. A number of possible values are defined but implementations may define additional values. Implementations will not return other values for error conditions described in this chapter. Each of the standard values listed below is described under the relevant error OM class to which it applies. The possible values are:

action-type-not-subscribed
asynchrony-not-supported
attribute-type-not-subscribed
authentication-error
bad-argument
bad-class
bad-session
busy
child-entry-specified
communications-problem
content-length-problem
content-type-problem
deferred-delivery-cancellation-rejected
delete-restriction-problem
eit-problem
element-of-service-not-subscribed
feature-not-negotiated
feature-unavailable
inappropriate-for-operation
inappropriate-matching
inconsistent-request
invalid-attribute-value
invalid-feature
invalid-parameters
message-submission-identifier-invalid
miscellaneous
no-such-class
no-such-entry
originator-invalid
out-of-memory
register-rejected
remote-bind-error
reversed
security
submission-control-violated
too-many-operations
too-many-sessions
unable-to-establish-association
unacceptable-security-context
unavailable
unavailable-action-type
unavailable-attribute-type
unsupported-critical-function
unwilling-to-perform

6.4 Attribute-Error

An instance of OM class **Attribute-Error** reports one or more attribute-related problems encountered while performing a function as requested on a particular occasion (see Section 9.2 in reference **X.413**.)

An instance of OM class **Attribute-Error** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attribute-Problems	Object(Attribute-Problem)	-	1/more	-

Table 6-2 OM Attributes of Attribute-Error

Attribute-Problems

This provides information about the attribute-related problem. An **Attribute-Error** can report several problems at a time.

6.5 Attribute-Problem

An instance of OM class **Attribute-Problem** documents one attribute-related problem encountered while performing a function as requested on a particular occasion (see Section 9.2 in reference **X.413**).

An instance of OM class **Attribute-Problem** has the OM attributes of its superclasses: *Object*, *Error* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Attribute-Type	String(Object-Identifier)	-	1	-
Attribute-Value	any	-	0-1	-

Table 6-3 OM Attributes of Attribute-Problem

Attribute-Type
This indicates the type of the attribute with which the problem is associated.

Attribute-Value
This indicates the value of the attribute with which the problem is associated.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is one of:

— **attribute-type-not-subscribed**
means that an attribute-type used as argument to the function is not one of those to which the client has subscribed.

— **inappropriate-for-operation**
means that an attribute-type used as an argument of the function is unsuitable for its required use.

— **inappropriate-matching**
means that the filter contains a filter-item in which an attribute is matched using an operation (equality, ordering or substrings) that is not defined for that attribute.

— **invalid-attribute-value**
means that a purported attribute-value specified as an argument of the function does not conform to the data-type for the attribute-type concerned.

— **unavailable-attribute-type**
means that a purported attribute-type used as an argument of the function is not one of those which is supported by the MS. If the MS is able to carry out the operation anyway, it is not prohibited from so doing.

6.6 Auto-Action-Request-Error

An instance of OM class **Auto-Action-Request-Error** reports one or more problems related to registration of an auto-action.

An instance of OM class **Auto-Action-Request-Error** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Auto-Action -Request-Problems	Object(Auto-Action -Request-Problem)	-	1/more	-

Table 6-4 OM Attributes of Auto-Action-Request-Error

Auto-Action-Request-Problems
This indicates type of problems that apply to the given auto-actions. An **Auto-Action-Request-Error** can report several problems at a time.

6.7 Auto-Action-Request-Problem

An instance of OM class **Auto-Action-Request-Problem** documents the type of problem encountered when attempting to register an auto-action.

An instance of OM class **Auto-Action-Request-Problem** has the OM attributes of its superclasses: *Object*, *Error* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Type	String(Object-Identifier)	-	1	-

Table 6-5 OM Attributes of Auto-Action-Request-Problem

Type
This identifies the auto-action type for which a problem is encountered when attempting to register it.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is one of:

— **action-type-not-subscribed**
means that an action-type used as argument to the function is not one of those to which the client has subscribed.

— **unavailable-action-type**
means that the action-type used as an argument of the function is not one of those supported by the MS.

6.8 Bind-Error

An instance of OM class **Bind-Error** indicates an error associated with the **Bind**( ) function.

An instance of OM class **Bind-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is one of:

— **authentication-error**
means that the **Bind**( ) function failed because the client's **Credentials** are not acceptable or are improperly specified.

— **unable-to-establish-association**
means that the MS has rejected the client's attempt to establish an association with the **Bind**( ) function.

— **unacceptable-security-context**
means that the **Bind**( ) function failed because the **Security-Context** proposed by the client for the **Bind**( ) function is unacceptable to the MS.

6.9 Cancel-Submission-Error

An instance of OM class **Cancel-Submission-Error** indicates a disruption in the performance of the **Cancel-Submission**( ) function.

An instance of OM class **Cancel-Submission-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*:

Problem

This identifies the problem. Its value is one of :

— **deferred-delivery-cancellation-rejected**
means that the MTS cannot cancel the deferred delivery of a message, either because the message has already been progressed for transfer and/or delivery, or because the MTS had provided the originator with proof-of-submission.

— **message-submission-identifier-invalid**
means that the deferred delivery of a message cannot be cancelled due to the invalid message-submission-identifier specified.

— **remote-bind-error**
means that the function requested cannot be performed because the MS is unable to bind to the MTS.

6.10 Communications-Error

An instance of OM class **Communications-Error** reports an error occurring in the other communications services supporting the MS. Such errors may include those arising in Remote Operation, Association Control, Presentation, Session and Transport.

An instance of OM class **Communications-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **communications-problem**
means that an error has occurred in the other communications services supporting the MS. Such errors include those arising in Remote Operation, Association Control, Presentation, Session and Transport.

6.11 Delete-Error

An instance of OM class **Delete-Error** reports one or more problems encountered while attempting to delete one or more entries from an information base.

An instance of OM class **Delete-Error** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Delete-Problems	Object(Delete-Problem)	-	1/more	-

Table 6-6 OM Attributes of Delete-Error

Delete-Problems

This provides information about problems encountered when attempting to delete one or more entries in an information base. Any number of individual problems may be indicated, each problem is reported along with the sequence-number of the entry causing the problem. A **Delete-Error** can report several problems at a time.

6.12 Delete-Problem

An instance of OM class **Delete-Problem** documents the type of problem encountered when attempting to delete an entry in an information base and also identifies the sequence-number of the entry.

An instance of OM class **Delete-Problem** has the OM attributes of its superclasses: *Object*, *Error* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Sequence-Number	Integer	-	1	-

Table 6-7 OM Attributes of Delete-Problem

Sequence-Number
This identifies the sequence-number of the entry causing the problem, **Problem**.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is one of:

— **child-entry-specified**
means that an attempt has been made to delete a child-entry.

— **delete-restriction-problem**
means that an attempt has been made to violate a restriction specified for the **Delete**( ) function.

6.13 Element-of-Service-Not-Subscribed-Error

An instance of OM class **Element-Of-Service-Not-Subscribed-Error** reports that the requested operation cannot be provided by the MTS because the MS has not subscribed to one of the elements of service the request requires.

An instance of OM class **Element-Of-Service-Not-Subscribed-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is:

— **element-of-service-not-subscribed**
means that the requested operation cannot be provided by the MTS because the MS has not subscribed to one of the elements of service the request requires.

6.14 Fetch-Restriction-Error

An instance of OM class **Fetch-Restriction-Error** reports an attempt to violate a restriction associated with the **Fetch**( ) function.

An instance of OM class **Fetch-Restriction-Error** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Fetch-Restriction -Problems	Object(Fetch-Restriction -Problem)	-	1/more	-

Table 6-8 OM Attributes of Fetch-Restriction-Error

Fetch-Restriction-Problems

This provides information about an attempt to violate a restriction associated with the **Fetch**( ) function. A **Fetch-Restriction-Error** can report several problems at a time.

6.15 Fetch-Restriction-Problem

An instance of OM class **Fetch-Restriction-Problem** reports on a Fetch restriction violation problem that was encountered.

An instance of OM class **Fetch-Restriction-Problem** has the OM attributes of its superclasses: *Object*, *Error* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Content-Type-Int	Integer †	-	0-1	-
Content-Type	String(Object-Identifier) †	-	0-1	-
EITs	String(Object-Identifier)	-	0/more	-
Content-Length	Integer	-	0-1	-

† For services based on MHS-1984 (see reference **MHS-1984**), only attribute "Content-Type-Int" is available. For services based on MHS-1988 (see reference **MHS-1988**), either "Content-Type-Int" or "Content-Type" is available.

Table 6-9 OM Attributes of Fetch-Restriction-Problem

Content-Type-Int
Content-Type
This indicates the content-type of the message with which the Fetch restriction problem is associated.

EIT
This indicates the encoded-information-types requested in the **Fetch**( ) function that are disallowed by the Fetch restrictions currently in effect.

Content-Length
This indicates the content-length of the message with which the Fetch restriction problem is associated.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is one of:

— **content-length-problem**
means that the content-length of the message being fetched exceeds that permitted by the Fetch restrictions currently in effect.

— **content-type-problem**
means that the content-type of the message being fetched is disallowed by the Fetch restrictions currently in effect.

— **eit-problem**
means that the encoded-information-types requested in the **Fetch**( ) function are disallowed by the Fetch restrictions currently in effect.

6.16 Inconsistent-Request-Error

An instance of OM class **Inconsistent-Request-Error** reports a problem where the requested operation cannot be provided by the MTS because the MS has made an inconsistent request.

An instance of OM class **Inconsistent-Request-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **inconsistent-request**
means that the requested operation cannot be provided by the MTS because the MS has made an inconsistent request.

6.17 Invalid-Parameters-Error

An instance of OM class **Invalid-Parameters-Error** indicates a semantic problem in the set of parameters received for a function. This error would be used, for example, to report that an optional parameter was present in the wrong context, or to report that a value for one of the parameters is inappropriate.

An instance of OM class **Invalid-Parameters-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **invalid-parameters**
means that there is a semantics problem in the set of parameters received for a function; this error would be used, for example, to report that an optional parameter was present in the wrong context, or to report that a value for one of the parameters is inappropriate.

6.18 Library-Error

An instance of OM class **Library-Error** reports an error detected by the interface function library.

An instance of OM class **Library-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Each function has several possible errors which can be detected by the interface library itself and which are returned directly by the function call. These errors occur when the library itself is capable of performing an action, submitting a service request or deciphering a response from the MS.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is one of:

— **asynchrony-not-supported**
means that the client requested the function be performed asynchronously when the service does not support asynchronous operations.

— **bad-argument**
means that a bad argument was supplied; e.g., using an instance of an OM class **Attribute** with no values of the OM attribute **Attribute-Value** as an input argument to a function will result in this error (since every MS attribute always has at least one value).

— **bad-class**
means that the OM class of an argument is not supported for this function.

— **bad-session**
means that there is no such session over which to present the function.

— **feature-unavailable**
means that either the function to be invoked or the OM class of some argument supplied is not supported by the service.

— **feature-not-negotiated**
means that either the functional unit(s) negotiated at **Initialize**( ) does include this function being invoked or the package(s) negotiated at **Initialize**( ) does include the OM class of some argument supplied.

— **miscellaneous**
means that a miscellaneous error has occurred. This error will be returned if the interface cannot clear a transient system error by retrying the affected system call.

— **no-such-class**
means that the argument supplied has a subobject which is an instance of a class not in the currently negotiated package(s).

— **out-of-memory**
means that no more memory can be allocated.

— **too-many-operations**
means that no more functions can be performed until at least one asynchronous operation has completed.

— **too-many-sessions** means that no more sessions can be bound with the message store until at least one existing session has unbound.

6.19 Originator-Invalid-Error

An instance of OM class **Originator-Invalid-Error** reports that the communique cannot be submitted because the originator was incorrectly identified.

An instance of OM class **Originator-Invalid-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **originator-invalid**
means that the communique cannot be submitted because the originator was incorrectly identified.

6.20 Range-Error

An instance of OM class **Range-Error** reports a problem related to the limit specified in a selector as an argument to a function.

An instance of OM class **Range-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **reversed**
means that the upper bound indicated a sequence-number or creation-time before that indicated by the lower bound.

6.21 Recipient-Improperly-Specified-Error

An instance of OM class **Recipient-Improperly-Specified-Error** gives the recipient names that were improperly specified in a communique presented for submission.

An instance of OM class **Recipient-Improperly-Specified-Error** has the OM attributes of its superclasses: *Object*, *Error* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Recipients	Object(OR-Name) †	-	1/more	-

† As defined in the X.400 API Specification (see reference **X.400**).

Table 6-10 OM Attributes of Recipient-Improperly-Specified-Error

Recipients

This lists the improperly specified recipients of the communique presented for submission.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is one of:

— **recipient-improperly-specified**
means that the communique cannot be submitted because it is directed to improperly specified recipients.

6.22 Register-Rejected-Error

An instance of OM class **Register-Rejected-Error** indicates that the **Register**( ) function has been rejected.

An instance of OM class **Register-Rejected-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **register-rejected**
means that the **Register**( ) function has been rejected.

6.23 Remote-Bind-Error

An instance of OM class **Remote-Bind-Error** reports that the requested function cannot be provided by the MS because the MS is unable to bind to the MTS.

An instance of OM class **Remote-Bind-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **remote-bind-error**
means that requested function cannot be provided by the MS because the MS is unable to bind to the MTS.

6.24 Security-Error

An instance of OM class **Security-Error** reports that the requested operation could not be provided by the MTS because it would violate the security policy in effect.

An instance of OM class **Security-Error** has the OM attributes of its superclasses: *Object*, *Error* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Security-Problem	Integer	-	1	-

Table 6-11 OM Attributes of Security-Error

Security-Problem
This identifies the cause of the violation of the security policy.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is:

— **security**
means that the requested operation could not be provided by the MTS because it would violate the security policy in effect.

6.25 Sequence-Number-Error

An instance of OM class **Sequence-Number-Error** reports one or more problems related to the sequence-numbers specified as argument to a function.

An instance of OM class **Sequence-Number-Error** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Sequence-Number -Problems	Object(Sequence-Number -Problem)	-	1/more	-

Table 6-12 OM Attributes of Sequence-Number-Error

Sequence-Number-Problems
This gives the sequence-numbers specified as argument to a function that caused whatever problems. A **Sequence-Number-Error** can report several problems at a time.

6.26 Sequence-Number-Problem

An instance of OM class **Sequence-Number-Problem** documents the type of problem related to sequence-numbers specified as argument to a function.

An instance of OM class **Sequence-Number-Problem** has the OM attributes of its superclasses: *Object*, *Error* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Sequence-Number	Integer	-	1	-

Table 6-13 OM Attributes of Sequence-Number-Problem

Sequence-Number
This identifies the sequence-number of the entry causing the problem.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is:

— **no-such-entry**
means that the sequence-number supplied does not match that of any entry in the information base.

6.27 Service-Error

An instance of OM class **Service-Error** reports a problem related to MS service.

An instance of OM class **Service-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem
This identifies the problem. Its value is one of:

— **busy**
means that the MS, or some part of it, is presently too busy to perform the requested function but may be able to do so after a short while.

— **unavailable**
means that the MS, or some part of it, is presently unavailable.

— **unwilling-to-perform**
means that the MS is not prepared to execute the requested function because it would lead to excessive consumption of resources.

6.28 Submission-Control-Violated-Error

An instance of OM class **Submission-Control-Violated-Error** reports the violation by the MS of a control on submission imposed by the MTS.

An instance of OM class **Submission-Control-Violated-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **submission-control-violated**
means that the MS has violated a control on submission imposed by the MTS.

6.29 System-Error

An instance of OM class **System-Error** reports an error occurring in the underlying operating system.

An instance of OM class **System-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is the same as that of *errno* defined in the C language.

The standard names of system errors are defined in the X/Open Portability Guide, Issue 4 (XPG4), System Interface and Headers (see reference **XSH**), and additional names may be defined by an implementation.

If a transient error occurs *[EINTR]* or *[EAGAIN]* implementations will retry the affected function and will not report these errors. If such an error persists, they may report the Library-Error (miscellaneous) or an implementation-defined library error.

6.30 Unsupported-Critical-Function-Error

An instance of OM class **Unsupported-Critical-Function-Error** reports that an argument of the function was marked as critical-for submission but is unsupported by the MTS.

An instance of OM class **Unsupported-Critical-Function-Error** has the OM attributes of its superclasses: *Object*, *Error* and no additional OM attributes.

Additional Constraints

The following OM attribute is inherited from the superclass *Error*.

Problem

This identifies the problem. Its value is:

— **unsupported-critical-function**
means that an argument of the function was marked as **critical-for-submission** but is unsupported by the MTS.

Chapter 7

MS General Attributes Class Definitions

7.1 Introduction

The MS standards define a number of MS attribute types, known as General-Attributes. This chapter lists the names for each of these MS attribute types and defines OM classes to represent those which are not represented directly by OM syntaxes. These constitute the **Message Store General Attributes Package**. The values of MS attributes are not restricted to those discussed in this chapter; new attribute types and syntaxes to be used in conjunction with an MS may be created in the future. Implementations are likely to add additional definitions. Section 3.4.1 on page 25 defines how the values of other syntaxes are represented in the interface.

The constants and OM classes defined in this chapter are additional to those in the MS Package (see Chapter 5 and Chapter 6), and they allow MS entries to be utilised.

This Message Store General Attributes Package is mandatory in that the service is required to provide those classes which can be supported by the underlying implementation of the MS (refer to Section 1.5 on page 7. For example, not all implementations of an MS support all the MS attributes defined in X.413 (see reference **X.413**). Therefore, the service need not support the OM classes corresponding to these unsupported MS attributes.

The object identifier associated with the Message Store General Attributes Package is represented by the constant:

Message-Store-General-Attributes-Package *{MS_GENERAL_ATTRIBUTES_PACKAGE}*

See Table 1-1 on page 5 for the value of this object identifier. The C constants associated with this package are given in the **<xmsga.h>** header.

The Object Management concepts and notation used in this chapter are introduced in Section 1.6 on page 9 of this document and are fully explained in the **XOM Specification** (see reference **XOM**). A complete explanation of the meaning of the MS attributes and object classes is not given since this is outside the scope of this specification (see reference **X.413**). The purpose here is simply to present the representation of these items in the interface.

7.2 MS Attribute Types

This section presents the MS attribute types defined in the Standards for use in MS entries. Each MS entry is composed of a number of MS attributes which comprises an attribute type along with one or more attribute values. The form of each value of an MS attribute is determined by the attribute syntax associated with the type of the attribute.

Note the distinction between MS attributes used with regard to the MS (see Section 1.3 on page 3) and OM attributes (see Section 1.6.3 on page 9). The term *MS attribute* or the unqualified term *attribute* is used to denote the MS construct, whereas the phrase *OM attribute* is used to denote the Object Management one.

In the interface, MS attributes appear as instances of the OM class **Attribute** with the attribute type represented as the value of the OM attribute **Attribute-Type** and the attribute value(s) represented as the value(s) of the OM attribute **Attribute-Values**. Each attribute type has an object identifier, assigned by the Standards, which is the value of the OM attribute **Attribute-Type**. These object identifiers are represented in the interface by constants with the same name as the MS attribute prefixed with A- for ease of identification (and correspondingly, the C variables begin with *MS_A_*).

This section contains two tables.

The first tabulates the names of the MS attribute types defined in the Standards, together with the object identifiers associated with each of them.

The second table gives the names of the MS attribute types together with the OM Value Syntax used in the interface to represent values of that MS attribute type. This OM Value Syntax is the syntax of the OM attribute **Attribute-Values**. The table also includes the range of lengths permitted for the string types, and an indication of whether the MS attribute can be multi-valued. Note that many of the OM Value Syntaxes are defined in the Message Handling Package of the X.400 Specification (see reference **X.400**).

Attribute Type	Object Identifier †
A-Child-Sequence-Numbers	0
A-Content	1
A-Content-Confidentiality-Algorithm-Identifier	2
A-Content-Correlator	3
A-Content-Identifier	4
A-Content-Integrity-Check	5
A-Content-Length	6
A-Content-Returned	7
A-Content-Type	8
A-Conversion-With-Loss-Prohibited	9
A-Converted-EITs	10
A-Creation-Time	11
A-Delivered-EITs	12
A-Delivery-Flags	13
A-DL-Expansion-History	14
A-Entry-Status	15
A-Entry-Type	16
A-Intended-Recipient-Name	17
A-Message-Delivery-Envelope	18
A-Message-Delivery-Identifier	19
A-Message-Delivery-Time	20
A-Message-Origin-Authentication-Check	21
A-Message-Security-Label	22
A-Message-Submission-Time	23
A-Message-Token	24
A-Original-EITs	25
A-Originator-Certificate	26
A-Originator-Name	27
A-Other-Recipient-Names	28
A-Parent-Sequence-Number	29
A-Per-Recipient-Report-Delivery-Fields	30
A-Priority	31
A-Proof-Of-Delivery-Request	32
A-Redirection-History	33
A-Report-Delivery-Envelope	34
A-Reporting-DL-Name	35
A-Reporting-MTA-Certificate	36
A-Report-Origin-Authentication-Check	37
A-Security-Classification	38
A-Sequence-Number	39
A-Subject-Submission-Identifier	40
A-This-Recipient-Name	41

† These object identifiers are preceded by:
{joint-iso-ccitt(2) mhs-motis(6) ms(4) attribute types(3)}

Table 7-1 Object Identifiers for MS Attribute Types

Attribute Type	OM Value Syntax	Value Length	(s)ingle/ (m)ulti -valued
A-Child-Sequence-Numbers	Integer	-	m
A-Content	Object(General-Content)	-	s
A-Content-Confidentiality-Algorithm-Id	String(Object-Identifier)	-	s
A-Content-Correlator	any	-	s
A-Content-Identifier	String(Printable)	-	s
A-Content-Integrity-Check	Object(Algorithm-And-Result)	-	s
A-Content-Length	Integer	-	s
A-Content-Returned	Boolean	-	s
A-Content-Type	String(Object-Identifier)	-	s
A-Conversion-With-Loss-Prohibited	Boolean	-	s
A-Converted-EITs	String(Object-Identifier)	-	m
A-Creation-Time	String(UTC-Time)	0-17	s
A-Delivered-EITs	String(Object-Identifier)	-	m
A-Delivery-Flags	Boolean	-	s
A-DL-Expansion-History	Object(Expansion-Record)	-	m
A-Entry-Status	Enum(Entry-Status)	-	s
A-Entry-Type	Enum(Entry-Type)	-	s
A-Intended-Recipient-Name	Object(OR-Name)	-	s
A-Message-Delivery-Envelope	Object(Delivery-Envelope)	-	s
A-Message-Delivery-Identifier	Object(MTS-Identifier)	-	s
A-Message-Delivery-Time	String(UTC-Time)	0-17	s
A-Message-Origin-Authentication-Check	Object(Algorithm-And-Result)	-	s
A-Message-Security-Label	Object(Security-Label)	-	s
A-Message-Submission-Time	String(UTC-Time)	0-17	s
A-Message-Token	Object(Asymmetric-Token)	-	s
A-Original-EITs	String(Object-Identifier)	-	m
A-Originator-Certificate	Object(Certificates)	-	s
A-Originator-Name	Object(OR-Name)	-	s
A-Other-Recipient-Names	Object(OR-Name)	-	m
A-Parent-Sequence-Number	Integer	-	s
A-Per-Recipient-Report-Delivery-Fields	Object(Delivered-Per-Recipient-Report)	-	m
A-Priority	Enum(Priority)	-	s
A-Proof-Of-Delivery-Request	Boolean	-	s
A-Redirection-History	Object(Redirection-Record)	-	m
A-Report-Delivery-Envelope	Object(Deliver-Report)	-	s
A-Reporting-DL-Name	Object(OR-Name)	-	s
A-Reporting-MTA-Certificate	Object(Certificates)	-	s
A-Report-Origin-Authentication-Check	Object(Algorithm-And-Result)	-	s
A-Security-Classification	Enum(Security-Classification)	-	s
A-Sequence-Number	Integer	-	s
A-Subject-Submission-Identifier	Object(MTS-Identifier)	-	s
A-This-Recipient-Name	Object(OR-Name)	-	s

Table 7-2 Value Syntax for MS Attribute Types

7.3 Class Hierarchy

This section depicts the hierarchical organisation of the classes defined in this chapter and thus shows which classes inherit additional OM attributes from their superclasses. Subclassification is indicated by indentation and the names of abstract classes are rendered in italics. Thus, for instance, the concrete class **Delivered-Per-Recipient-NDR** is an immediate subclass of the abstract class *Delivered-Per-Recipient-Report* which in turn is an immediate subclass of the abstract class *Object*.

Object (defined in the XOM Specification)

— Algorithm (defined in the X.400 Specification)

  — Algorithm-And-Result (defined in the X.400 Specification)

— Asymmetric-Token (defined in the X.400 Specification)

— Certificates (defined in the XDS Specification)

— General-Content (defined in the X.400 Specification)

— Delivered-Report (defined in the X.400 Specification)

— Expansion-Record (defined in the X.400 Specification)

— MTS-Identifier (defined in the X.400 Specification)

— OR-Name (defined in the X.400 Specification)

— *Delivered-Per-Recipient-Report* (defined in the X.400 Specification)

  — Delivered-Per-Recipient-DR (defined in the X.400 Specification)

  — Delivered-Per-Recipient-NDR (defined in the X.400 Specification)

— Redirection-Record (defined in the X.400 Specification)

— Security-Label (defined in the X.400 Specification)

7.4 Syntax Definitions

This section defines the MS class enumeration syntaxes, i.e., the syntaxes in the Enumeration group specific to MS General Attributes.

7.4.1 Entry-Status

An instance of the enumeration syntax **Entry-Status** indicates the current processing status of an MS entry. Its value is chosen from the following:

- **new**

  The message has neither been accessed via the **List**( ) function nor has it been automatically processed by the MS.

- **listed**

  The message has been accessed via the **List**( ) function or the **Fetch**( ) function but it has yet to be completely ''processed''.

- **processed**

  The message has been ''completely fetched'' or the MS has performed some auto action on it. The exact definition of ''completely fetched'' is content-specific and is defined by the corresponding content-specific standards. The Entry-Status of a (non)-delivery notification becomes **processed** when the delivered report envelope is retrieved.

7.4.2 Entry-Type

An instance of the enumeration syntax **Entry-Type** indicates the type of an MS entry. Its value is chosen from the following:

- **delivered-message**
- **delivered-report**
- **returned-content**

7.4.3 Priority

An instance of the enumeration syntax **Priority** indicates the priority of the delivered message. Its value is chosen from the following:

- **normal**
- **low**
- **urgent**

7.4.4 Security-Classification

An instance of the enumeration syntax **Security-Classification** indicates security classification of a message. Its value is chosen from the following:

- **unmarked**
- **unclassified**
- **restricted**
- **confidential**
- **secret**
- **top-secret**

Chapter 8

MS IM Attributes Class Definitions

8.1 Introduction

The MS interface may be used with MS attributes other than those in the mandatory Message Store General Attributes Package described in Chapter 7. Other attribute types and syntaxes may be used in conjunction with an MS. This chapter describes one such optional package, the *Message Store Interpersonal Messaging (MS IM) Attributes Package*.

This *MS IM Attributes Package* contains all definitions for the Interpersonal Messaging MS attributes as defined in the standards (see Annex J of reference **X.420**). Note that these attributes referred to as the Interpersonal Messaging MS Attributes in the X.420 will be known as the MS IM Attributes in this specification.

This chapter lists the names for each of these MS IM attribute types and defines OM classes to represent those which are not represented directly by OM syntaxes.

The *MS IM Attributes Package* is optional. The constants and OM classes defined in this chapter are additional to those in the MS Package (see Chapter 5 and Chapter 6), and they are not essential to the working of the interface but instead allow the access of the MS IM attributes.

The object identifier associated with the MS IM Attributes Package is represented by the constant:

Message-Store-Interpersonal-Messaging-Attributes-Package
{MS_IM_ATTRIBUTES_PACKAGE}.

See Table 1-1 on page 5 for the value of this object identifier. The C constants associated with this package are given in the **<xmsima.h>** header.

The Object Management concepts and notation used in this chapter are introduced in Section 1.6 on page 9 of this document and are fully explained in the XOM Specification (see reference **XOM**). A complete explanation of the meaning of the MS IM attributes and object classes is not given since this is outside the scope of this specification (for details, see reference **X.420**). The purpose here is simply to present the representation of these items in the interface.

8.2 MS Interpersonal Messaging Attribute Types

This section presents the MS Interpersonal Messaging attribute types defined in the X.420 standards (see reference **X.420**) for use in MS entries. Each MS entry is composed of a number of MS attributes which comprise an attribute type along with one or more attribute values. The form of each value of an MS IM attribute is determined by the attribute syntax associated with the type of the attribute.

Note the distinction between MS attributes used with regard to the MS (see Section 1.3 on page 3 and OM attributes (see Section 1.6.3 on page 9. The term *MS attribute* or the unqualified term *attribute* is used to denote the MS construct, whereas the phrase *OM attribute* is used to denote the Object Management one.

In the interface, MS IM attributes appear as instances of the OM class **Attribute** with the attribute type represented as the value of the OM attribute **Attribute-Type** and the attribute value(s) represented as the value(s) of the OM attribute **Attribute-Values**. Each attribute type has an object identifier, assigned by the Standards, which is the value of the OM attribute **Attribute-Type**. These object identifiers are represented in the interface by constants with the same name as the MS IM attribute prefixed with IM- for ease of identification (and correspondingly, the C variables begin with *MS_IM_*).

This section contains two tables.

The first tabulates the names of the MS IM attribute types defined in Annex J of X.420 (see reference **X.420**), together with their respective object identifiers.

The second table gives the names of the MS IM attribute types together with the OM Value Syntax used in the interface and the range of lengths permitted for the string types, and an indication of whether the MS attribute can be multi-valued. The OM Value Syntax is the syntax of the OM attribute **Attribute-Values**. Note that many of the OM Value Syntaxes are defined in the Interpersonal Messaging Package of the X.400 API Specification (see reference **X.400**).

Table 8-1 Object Identifiers for MS Interpersonal Messaging Attribute Types

† These object identifiers are preceded by:
{joint-iso-ccitt(2) mhs-motis(6) ipms(1)}

MS IM Attributes	Object Identifier †
IM-Acknowledgment-Mode	9,9
IM-Authorizing-Users	7,10
IM-Auto-Forward-Comment	9,6
IM-Auto-Forwarded	7,9
IM-Bilaterally-Defined-Body-Parts	8,10
IM-Blind-Copy-Recipients	7,13
IM-Body	8,0
IM-Conversion-EITs	9,3
IM-Copy-Recipients	7,12
IM-Discard-Reason	9,5
IM-Expiry-Time	7,5
IM-Extended-Body-Part-Types	8,12
IM-G3-Fax-Body-Parts	8,3
IM-G3-Fax-Data	8,22

MS IM Attributes	Object Identifier †
IM-G3-Fax-Parameters	8,15
IM-G4-Class1-Body-Parts	8,4
IM-Heading	7,0
IM-IA5-Text-Body-Parts	8,1
IM-IA5-Text-Data	8,20
IM-IA5-Text-Parameters	8,13
IM-Importance	7,7
IM-Incomplete-Copy	7,17
IM-IPM-Entry-Type	6,0
IM-IPM-Preferred-Recipient	9,2
IM-IPM-Synopsis	6,1
IM-IPN-Originator	9,1
IM-Languages	7,18
IM-Message-Body-Parts	8,8
IM-Message-Data	8,26
IM-Message-Parameters	8,19
IM-Mixed-Mode-Body-Parts	8,9
IM-Nationally-Defined-Body-Parts	8,11
IM-Non-Receipt-Reason	9,4
IM-NRN-Requestors	7,20
IM-Obsoleted-IPMs	7,14
IM-Originator	7,2
IM-Primary-Recipients	7,11
IM-Receipt-Time	9,8
IM-Related-IPMs	7,15
IM-Replied-To-IPM	7,3
IM-Reply-Recipients	7,16
IM-Reply-Requestors	7,21
IM-Reply-Time	7,6
IM-Returned-IPM	9,7
IM-RN-Requestors	7,19
IM-Sensitivity	7,8
IM-Subject	7,4
IM-Subject-IPM	9,0
IM-Suppl-Receipt-Info	9,10
IM-Teletex-Body-Parts	8,5
IM-Teletex-Data	8,23
IM-Teletex-Parameters	8,16
IM-This-IPM	7,1
IM-Videotex-Body-Parts	8,6
IM-Videotex-Data	8,24
IM-Videotex-Parameters	8,17

Table 8-2 Value Syntax for MS Interpersonal Messaging Attribute Types

MS IM Attributes	OM Value Syntax	Value Length	(s)ingle/ (m)ulti -valued
IM-Acknowledgment-Mode	Enum(Acknowledgment-Mode)	-	s
IM-Authorizing-Users	Object(OR-Descriptor)	-	m
IM-Auto-Forward-Comment	String(Printable)	0-256	s
IM-Auto-Forwarded	Boolean	-	s
IM-Bilaterally-Defined-Body-Parts	Object(Bil-Defined-Body-Part)	-	m
IM-Blind-Copy-Recipients	Object(Recipient-Specifier)	-	m
IM-Body	Object(Body)	-	s
IM-Conversion-EITs	String(Object-Identifier)	-	m
IM-Copy-Recipients	Object(Recipient-Specifier)	-	m
IM-Discard-Reason	Enum(Discard-Reason)	-	s
IM-Expiry-Time	String(UTC-Time)	0-17	s
IM-Extended-Body-Part-Types	String(Object Identifier)	-	m
IM-G3-Fax-Body-Parts	Object(G3-Fax-Body-Part)	-	m
IM-G3-Fax-Data	Object(G3-Fax-Data)	-	m
IM-G3-Fax-Parameters	Object(G3-Fax-NBPs)	-	s
IM-G4-Class1-Body-Parts	Object(G4-Class1-Body-Part)	-	m
IM-Heading	Object(Heading)	-	s
IM-IA5-Text-Body-Parts	Object(IA5-Text-Body-Part)	-	m
IM-IA5-Text-Data	String(IA5)	-	m
IM-IA5-Text-Parameters	Enum(IA5-Repertoire)	-	m
IM-Importance	Enum(Importance)	-	s
IM-Incomplete-Copy	Boolean	-	s
IM-IPM-Entry-Type	Enum(IPM-Entry-Type)	-	s
IM-IPM-Preferred-Recipient	Object(OR-Descriptor)	-	s
IM-IPM-Synopsis	Object(IPM-Synopsis)	-	s
IM-IPN-Originator	Object(OR-Descriptor)	-	s
IM-Languages	String(Printable)	2-5	m
IM-Message-Body-Parts	Object(Message-Body-Part)	-	m
IM-Message-Data	Object(Interpersonal-Message)	-	s
IM-Message-Parameters	Object(Delivery-Envelope)	-	s
IM-Mixed-Mode-Body-Parts	Object(Mixed-Mode-Body-Part)	-	m
IM-Nationally-Defined-Body-Parts	Object(Nat-Defined-Body-Part)	-	m
IM-Non-Receipt-Reason	Enum(Non-Receipt-Reason)	-	s
IM-NRN-Requestors	Object(OR-Descriptor)	-	m
IM-Obsoleted-IPMs	Object(IPM-Identifier)	-	m
IM-Originator	Object(OR-Descriptor)	-	s
IM-Primary-Recipients	Object(Recipient-Specifier)	-	m
IM-Receipt-Time	String(UTC-Time)	0-17	s
IM-Related-IPMs	Object(IPM-Identifier)	-	m
IM-Replied-To-IPM	Object(IPM-Identifier)	-	s
IM-Reply-Recipients	Object(OR-Descriptor)	-	m
IM-Reply-Requestors	Object(OR-Descriptor)	-	m
IM-Reply-Time	String(UTC-Time)	0-17	s
IM-Returned-IPM	Object(IPM-Identifier)	-	s
IM-RN-Requestors	Object(OR-Descriptor)	-	m
IM-Sensitivity	Enum(Sensitivity)	-	s
IM-Subject	String(Teletex)	0-128	s

MS IM Attributes	OM Value Syntax	Value Length	(s)ingle/ (m)ulti -valued
IM-Subject-IPM	Object(IPM-Identifier)	-	s
IM-Suppl-Receipt-Info	String(Printable)	1-256	s
IM-Teletex-Body-Parts	Object(Teletex-Body-Part)	-	m
IM-Teletex-Data	Object(Teletex-Data)	-	m
IM-Teletex-Parameters	Object(Teletex-Parameters)	-	m
IM-This-IPM	Object(IPM-Identifier)	-	s
IM-Videotex-Body-Parts	Object(Videotex-Body-Part)	-	m
IM-Videotex-Data	String(Videotex)	-	m
IM-Videotex-Parameters	Enum(Videotex-Syntax)	-	m

8.3 Class Hierarchy

This section depicts the hierarchical organisation of the classes defined in this chapter and thus shows which classes inherit additional OM attributes from their superclasses. Subclassification is indicated by indentation and the names of abstract classes are rendered in italics. Thus, for instance, the concrete class **Bilaterally-Defined-Body-Part** is an immediate subclass of the abstract class *Body-Part* which in turn is an immediate subclass of the abstract class *Object*.

Object (defined in the XOM Specification - see reference **XOM**)

— *Body-Part* (defined in the X.400 API Specification - see reference **X.400**)

- — Bilaterally-Defined-Body-Part (defined in the X.400 API Specification)
- — Body (defined in the X.400 API Specification)
- — G3-Fax-Body-Part (defined in the X.400 API Specification)
- — G4-Class1-Body-Part (defined in the X.400 API Specification)
- — IA5-Text-Body-Part (defined in the X.400 API Specification)
- — Message-Body-Part (defined in the X.400 API Specification)
- — Mixed-Mode-Body-Part (defined in the X.400 API Specification)
- — Nationally-Defined-Body-Part (defined in the X.400 API Specification)
- — Teletex-Body-Part (defined in the X.400 API Specification)
  - — Teletex-Data
  - — Teletex-Parameters
- — Videotex-Body-Part (defined in the X.400 API Specification)
- — Body-Part-Synopsis
- — *Content* (defined in the X.400 API Specification)
  - — Interpersonal-Message (defined in the X.400 API Specification)
  - — *Interpersonal-Notification* (defined in the X.400 API Specification)
    - — Non-Receipt-Notification (defined in the X.400 API Specification)
    - — Receipt-Notification (defined in the X.400 API Specification)
- — Extensible-Object (defined in the X.400 API Specification)
  - — Delivery-Envelope (defined in the X.400 API Specification)
- — G3-Fax-Data (defined in the X.400 API Specification)
- — G3-Fax-NBPs (defined in the X.400 API Specification)
- — Heading
- — IPM-Identifier (defined in the X.400 API Specification)
- — IPM-Synopsis
- — Message-Body-Part-Synopsis

— Non-Message-Body-Part-Synopsis

— OR-Descriptor (defined in the X.400 API Specification)

— Recipient-Specifier (defined in the X.400 API Specification)

8.4 Body

An instance of OM class **Body** gives all the body parts of a message.

An instance of OM class **Body** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Body-Part	Object(Body-Part)	-	0/more	-

Table 8-3 OM Attributes of Body

Body-Part
A body part of the message.

8.5 Body-Part-Synopsis

An instance of OM class **Body-Part-Synopsis** gives the synopsis for an individual body part.

An instance of OM class **Body-Part-Synopsis** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Message-Body-Part-Synopsis	Object(Message-Body-Part-Synopsis)	-	0-1	-
Non-Message-Body-Part-Synopsis	Object(Non-Message-Body-Part-Synopsis)	-	0-1	-

Table 8-4 OM Attributes of Body-Part-Synopsis

Note: No instance will contain more than one of the above OM attributes.

Message-Body-Part-Synopsis
This is the synopsis of a body part that is of type Message.

Non-Message-Body-Part-Synopsis
This is gives the synopsis of a body part that is of type other than Message.

8.6 G3-Fax-Data

An instance of OM class **G3-Fax-Data** gives the image data of a G3 facsimile body part of an interpersonal message.

An instance of OM class **G3-Fax-Data** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Images	String(Bit)	-	0/more	-

Table 8-5 OM Attributes of G3-Fax-Data

Images
The G3 facsimile images.

8.7 Heading

An instance of OM class **Heading** gives all the heading fields of an interpersonal message.

An instance of OM class **Heading** has the OM attributes of its superclasses: *Object*, and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
This-IPM	Object(IPM-Identifier)	-	1	-
Originator	Object(OR-Descriptor)	-	0-1	-
Authorizing-Users	Object(OR-Descriptor)	-	0/more	-
Primary-Recipients	Object(Recipient-Specifier)	-	0/more	-
Copy-Recipients	Object(Recipient-Specifier)	-	0/more	-
Blind-Copy-Recipients	Object(Recipient-Specifier)	-	0/more	-
Replied-To-IPM	Object(IPM-Identifier)	-	0-1	-
Obsoleted-IPMs	Object(IPM-Identifier)	-	0/more	-
Related-IPMs	Object(IPM-Identifier)	-	0/more	-
Subject	String(Teletex)	0-128	0-1	-
Expiry-Time	String(UTC-Time)	0-17	0-1	-
Reply-Time	String(UTC-Time)	0-17	0-1	-
Reply-Recipients	Object(OR-Descriptor)	-	0/more	-
Importance	Enum(Importance)	-	1	normal
Sensitivity	Enum(Sensitivity)	-	0-1	-
Auto-Forwarded	Boolean	-	1	false
Extensions	Object(Attribute)	-	0/more	-

Table 8-6 OM Attributes of Heading

This-IPM
This identifies the interpersonal message.

Originator
This identifies the originator of the interpersonal message.

Authorizing-Users
This identifies zero or more users who authorised the origination of the interpersonal message.

Primary-Recipients
This identifies zero or more users and distribution lists who are the ‘‘primary recipients’’ of the interpersonal message.

Copy-Recipients
This identifies zero or more users and distribution lists who are the ‘‘copy recipients’’ of the interpersonal message.

Blind-Copy-Recipients
This identifies zero or more users and distribution lists who are the ‘‘blind copy recipients’’ of the interpersonal message. A blind copy recipient is one whose role is not disclosed to primary and copy recipients.

Replied-To-IPM
This identifies the interpersonal message to which the present interpersonal message is a reply.

Obsoleted-IPMs
This identifies the interpersonal messages that the authorising users consider

the present interpersonal message to obsolete.

Related-IPMs
This identifies the interpersonal messages that the authorising users consider related to the present interpersonal message.

Subject
This identifies the subject of the interpersonal message.

Expiry-Time
This identifies when the authorising users consider the interpersonal message to lose its validity.

Reply-Time
This identifies when the authorising users request (but do not demand) that any replies to the present interpersonal message be originated.

Reply-Recipients
This identifies zero or more users or distribution lists whom the authorising users request (but do not demand) be among the preferred recipients of any replies to the present interpersonal message.

Importance
This identifies the importance (low, normal or high) that the authorising users attach to the interpersonal message.

Sensitivity
This identifies the sensitivity (personal, private or company-confidential) that the authorising users attribute to the interpersonal message.

Auto-Forwarded
This indicates whether the interpersonal message is a result of auto-forwarding.

Extensions
This conveys information accommodated by no other heading field. Some extensions (e.g., Languages, Incomplete-Copy) are defined in X.420, Annex A (see reference **X.420**).

8.8 IPM-Synopsis

An instance of OM class **IPM-Synopsis** describes the structure, characteristics, size and processing status of an interpersonal message at the granularity of individual body parts.

An instance of OM class **IPM-Synopsis** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Body-Part-Synopsis	Object(Body-Part-Synopsis)	-	0/more	-

Table 8-7 OM Attributes of IPM-Synopsis

Body-Part-Synopsis
This is synopsis for an individual body part.

8.9 Message-Body-Part-Synopsis

An instance of OM class **Message-Body-Part-Synopsis** gives the synopsis of a body part that is of type Message.

An instance of OM class **Message-Body-Part-Synopsis** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Sequence-Number	Integer	-	1	-
Synopsis	Object(IPM-Synopsis)	-	1	-

Table 8-8 OM Attributes of Message-Body-Part-Synopsis

Sequence-Number
This is the sequence number assigned by the MS to the entry that the Message body part represents.

Synopsis
This is the synopsis of the interpersonal message that forms the content of the message that contains the body part.

8.10 Non-Message-Body-Part-Synopsis

An instance of OM class **Non-Message-Body-Part-Synopsis** gives the synopsis of a body part that is of type other than Message.

An instance of OM class **Non-Message-Body-Part-Synopsis** has the OM attributes of its superclasses: *Object* and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Type	String(Object-Identifier)	-	1	-
Parameters	Object(External) †	-	1	-
Size	Integer	-	1	-
Processed	Boolean	-	1	false

† As defined in the XOM Specification (see reference **XOM**).

Table 8-9 OM Attributes of Non-Message-Body-Part-Synopsis

Type

This is the extended type of the body part (i.e., the Direct-reference component of its Data component).

Parameters

This is the format and control parameters of the body part (i.e., its Parameters component).

Size

This is size in octets of the encoding of the Encoding component of the body part's Data component when the BER of X.209 (see reference **X.209**) are followed.

Processed

This indicates whether or not the body part has been conveyed to the UA by means of the **List**( ) or the **Fetch**( ) function.

8.11 Teletex-Data

An instance of OM class **Teletex-Data** gives the data of a Teletex body part of an interpersonal message.

An instance of OM class **Teletex-Data** has the OM attributes of its superclasses: *Object* and additionally the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Teletex-Document	String(Teletex)	-	0/more	-

Table 8-10 OM Attributes of Teletex-Data

Teletex-Document
A page of the Teletex document.

8.12 Teletex-Parameters

An instance of OM class **Teletex-Parameters** describes the parameters of a Teletex body part of an interpersonal message.

An instance of OM class **Teletex-Parameters** has the OM attributes of its superclasses: *Object*, **Teletex-Body-Part** (defined in the IM Package in the X.400 API Specification - see reference **X.400**) and additionally, the OM attributes listed below.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Number-Of-Pages	Integer	-	0-1	-

Table 8-11 OM Attributes of Teletex-Parameters

Number-Of-Pages
This indicates the number of pages contained in a Teletex body part of an interpersonal message.

Additional Constraints:

The inherited attribute **Teletex-Document** shall not be present.

8.13 Syntax Definitions

This section defines the MS IM class enumeration syntaxes, i.e., the syntaxes in the Enumeration group specific to Interpersonal Messaging MS attributes.

8.13.1 Acknowledgment-Mode

An instance of the enumeration syntax **Acknowledgment-Mode** identifies how a report notification may be originated. Its value is chosen from the following:

- **automatic**
  The originator authorises report notifications in a blanket manner.
- **manual**
  The originator authorises report notifications individually.

8.13.2 Discard-Reason

An instance of the enumeration syntax **Discard-Reason** indicates why an IPM may be discarded. Its value is chosen from the following:

- **ipm-expired**
  The time identified by the IPM's Expiry-Time attribute arrives and expired IPMs are being discarded.
- **ipm-obsoleted**
  The Obsoleted-IPMs attribute of another IPM delivered to the recipient identifies the IPM and obsolete IPMs are being discarded.
- **no-discard**
  The IPM is not discarded.
- **user-terminated**
  The recipient's IM subscription is terminated.

8.13.3 IA5-Repertoire

An instance of the enumeration syntax **IA5-Repertoire** identifies the character set to which the text portion of an IA5 Text Body Part is constrained. Its value is chosen from the following:

- **IA5**
  The full IA5 character set (which is similar to ASCII).
- **ITA2**
  The ITA2 (i.e., Telex) character set.

8.13.4 Importance

An instance of the enumeration syntax **Importance** identifies the importance that an IM's authorising users may attach to the IM. Its value is chosen from the following: high, low or routine. These values are not defined by this document but are given meaning by users.

8.13.5 IPM-Entry-Type

An instance of the enumeration syntax **IPM-Entry-Type** identifies the type of Interpersonal Messaging entry. Its value is chosen from the following:

IPM The entry is an interpersonal message.

RN The entry is a receipt notification.

NRN The entry is a non-receipt notification.

8.13.6 Non-Receipt-Reason

An instance of the enumeration syntax **Non-Receipt-Reason** indicates why a user may not receive an IM after its delivery to him. Its value is chosen from the following:

- **ipm-auto-forwarded**
  The IPM is automatically forwarded.
- **ipm-discarded**
  The IPM is discarded.

8.13.7 Sensitivity

An instance of the enumeration syntax **Sensitivity** indicates how sensitive an IM's authorising users may consider the IM to be. Its value is chosen from the following:

- **company-confidential**
  The IM should be handled according to company-defined procedures for confidential information.
- **not-sensitive**
  The IM is not sensitive.
- **personal**
  The IM is addressed to its intended recipients as individuals, not as professionals.
- **private**
  The IM should be disclosed to no other than its intended recipients.

8.13.8 Videotex-Syntax

An instance of the enumeration syntax **Videotex-Syntax** identifies the syntax of the data portion of a Videotex body part. Its value is chosen from the following:

data-syntax-1 Data syntax 1 as defined by T.100.

data-syntax-2 Data syntax 2 as defined by T.100.

data-syntax-3 Data syntax 3 as defined by T.100.

ids The IDS syntax as defined by T.100.

unspecified The syntax is unspecified.

Chapter 9

Headers

This chapter gives the symbols which are defined in these C headers:

<xms.h>
<xmsga.h>
<xmsima.h>

Where the values of the symbols are indicated, the values are an integral part of the interface. Where a value is not given, the value on a particular system will be determined by the vendor or by an administrator.

9.1 <xms.h>

The **<xms.h>** defines *MS_PACKAGE*, the constant value for the Message Store package. It also declares the interface functions, the data structures passed to and from those functions and the defined constants used by the functions and data structures for the Message Store package.

All client programs that include this header must first include the Object Management header **<xom.h>** (see reference **XOM**).

All object identifiers are represented by constants defined in the header. These constants are used with the macros defined in the XOM API Specification.

```
/*
 *  xms.h        (Message Store Package)
 */

#ifndef XMS_HEADER
#define XMS_HEADER


/* MS Package object identifier */

#define OMP_O_MS_PACKAGE        "\x56\x06\x01\x02\x06\x01"


/* Type Definitions */

typedef OM_private_object       MS_status;

typedef OM_sint                 MS_invoke_id;

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

```
/* Interface Functions - Function Prototypes */

MS_status  ms_bind (
            OM_object            bind_argument,
            OM_private_object    *bind_result_return,
            OM_private_object    *bound_session_return
);

MS_status  ms_cancel_submission  (
            OM_private_object    session,
            OM_object            mts_identifier,
            MS_invoke_id         *invoke_id_return
);

MS_status  ms_check_alert  (
            OM_private_object    session,
            OM_private_object    *check_alert_result_return,
);

MS_status  ms_delete  (
            OM_private_object    session,
            OM_object            delete_argument,
            MS_invoke_id         *invoke_id_return
);

MS_status  ms_fetch  (
            OM_private_object    session,
            OM_object            fetch_argument,
            OM_private_object    *fetch_result_return,
            MS_invoke_id         *invoke_id_return
);

MS_status  ms_initialize  (
            MS_feature           feature_list[],
            OM_sint              *max_sessions,
            OM_sint              *max_outstanding_operations,
            OM_workspace         *workspace_return,
);

MS_status  ms_list  (
            OM_private_object    session,
            OM_object            list_argument,
            OM_private_object    *list_result_return,
            MS_invoke_id         *invoke_id_return
);
```

```
MS_status  ms_receive_result (
            OM_private_object    session,
            OM_uint              *completion_flag_return,
            MS_status            *operation_status_return,
            OM_private_object    *result_return,
            MS_invoke_id         *invoke_id_return
);

MS_status  ms_register (
            OM_private_object    session,
            OM_object            register_argument,
            MS_invoke_id         *invoke_id_return
);

MS_status  ms_register_ms(
            OM_private_object    session,
            OM_object            register_ms_argument,
            MS_invoke_id         *invoke_id_return
);

void  ms_shutdown (
            void
);

MS_status  ms_submit (
            OM_private_object    session,
            OM_object            communique,
            OM_private_object    *submission_results_return,
            MS_invoke_id         *invoke_id_return
);

MS_status  ms_summarize (
            OM_private_object    session,
            OM_object            summarize_argument,
            OM_private_object    *summarize_result_return,
            MS_invoke_id         *invoke_id_return
);

MS_status  ms_unbind (
            OM_private_object    session
);

MS_status  ms_wait (
            OM_private_object    session,
            OM_uint32            interval,
            OM_private_object    *wait_result_return,
            MS_invoke_id         *invoke_id_return
);
```

```
/* Classes */

/*
 * Note: Every client program must explicitly import into
 * every compilation unit (C source program) the classes or
 * Object Identifiers that it uses. Each of these classes or
 * Object Identifier names must then be explicitly exported from
 * just one compilation unit.
 * Importing and exporting can be done using the OM_IMPORT and
 * OM_EXPORT macros respectively (see the XOM API Specification).
 * For instance, the client program uses
 *                OM_IMPORT( MS_C_ATTRIBUTE_DEFAULTS )
 * which in turn will make use of
 *                OMP_O_MS_C_ATTRIBUTE_DEFAULTS
 * defined below.
 */

#define OMP_O_MS_C_ALERT_ADDRESS               "\x56\x06\x01\x02\x06\x01\x01"
#define OMP_O_MS_C_ATTRIBUTE_DEFAULTS          "\x56\x06\x01\x02\x06\x01\x02"
#define OMP_O_MS_C_ATTRIBUTE_ERROR             "\x56\x06\x01\x02\x06\x01\x03"
#define OMP_O_MS_C_ATTRIBUTE_PROBLEM           "\x56\x06\x01\x02\x06\x01\x04"
#define OMP_O_MS_C_ATTRIBUTE_SELECTION         "\x56\x06\x01\x02\x06\x01\x05"
#define OMP_O_MS_C_AUTO_ACTION                 "\x56\x06\x01\x02\x06\x01\x06"
#define OMP_O_MS_C_AUTO_ACTION_DEREGISTRATION  "\x56\x06\x01\x02\x06\x01\x07"
#define OMP_O_MS_C_AUTO_ACTION_REGISTRATION    "\x56\x06\x01\x02\x06\x01\x08"
#define OMP_O_MS_C_AUTO_ACTION_REQUEST_ERROR   "\x56\x06\x01\x02\x06\x01\x09"
#define OMP_O_MS_C_AUTO_ACTION_REQUESTPROBLEM  "\x56\x06\x01\x02\x06\x01\x0A"
#define OMP_O_MS_C_AUTO_ALERT_REG_PARAMETERS   "\x56\x06\x01\x02\x06\x01\x0B"
#define OMP_O_MS_C_AUTO_FORWARD_ARGUMENTS      "\x56\x06\x01\x02\x06\x01\x0C"
#define OMP_O_MS_C_AUTO_FORWARD_REG_PARAMETERS "\x56\x06\x01\x02\x06\x01\x0D"
#define OMP_O_MS_C_BIND_ARGUMENT               "\x56\x06\x01\x02\x06\x01\x0E"
#define OMP_O_MS_C_BIND_ERROR                  "\x56\x06\x01\x02\x06\x01\x0F"
#define OMP_O_MS_C_BIND_RESULT                 "\x56\x06\x01\x02\x06\x01\x10"
#define OMP_O_MS_C_CANCEL_SUBMISSION_ERROR     "\x56\x06\x01\x02\x06\x01\x11"
#define OMP_O_MS_C_CHANGE_CREDENTIALS          "\x56\x06\x01\x02\x06\x01\x12"
#define OMP_O_MS_C_CHECK_ALERT_RESULT          "\x56\x06\x01\x02\x06\x01\x13"
#define OMP_O_MS_C_COMMON_CONTROLS             "\x56\x06\x01\x02\x06\x01\x14"
#define OMP_O_MS_C_COMMUNICATIONS_ERROR        "\x56\x06\x01\x02\x06\x01\x15"
#define OMP_O_MS_C_CREATION_TIME_RANGE         "\x56\x06\x01\x02\x06\x01\x16"
#define OMP_O_MS_C_CREDENTIALS                 "\x56\x06\x01\x02\x06\x01\x17"
#define OMP_O_MS_C_DEFAULT_DELIVERY_CONTROLS   "\x56\x06\x01\x02\x06\x01\x18"
#define OMP_O_MS_C_DELETE_ARGUMENT             "\x56\x06\x01\x02\x06\x01\x19"
#define OMP_O_MS_C_DELETE_ERROR                "\x56\x06\x01\x02\x06\x01\x1A"
#define OMP_O_MS_C_DELETE_PROBLEM              "\x56\x06\x01\x02\x06\x01\x1B"
#define OMP_O_MS_C_DELIVERABLE_CONTENT_TYPES   "\x56\x06\x01\x02\x06\x01\x1C"
#define OMP_O_MS_C_ELMT_NOT_SUBSCRIBED_ERROR   "\x56\x06\x01\x02\x06\x01\x1D"
#define OMP_O_MS_C_ERROR                       "\x56\x06\x01\x02\x06\x01\x1E"
#define OMP_O_MS_C_FETCH_ARGUMENT              "\x56\x06\x01\x02\x06\x01\x1F"
#define OMP_O_MS_C_FETCH_ATTRIBUTE_DEFAULTS    "\x56\x06\x01\x02\x06\x01\x20"
#define OMP_O_MS_C_FETCH_RESTRICTION_ERROR     "\x56\x06\x01\x02\x06\x01\x21"
#define OMP_O_MS_C_FETCH_RESTRICTION_PROBLEM   "\x56\x06\x01\x02\x06\x01\x22"
#define OMP_O_MS_C_FETCH_RESULT                "\x56\x06\x01\x02\x06\x01\x23"
#define OMP_O_MS_C_INCONSISTENT_REQUEST_ERROR  "\x56\x06\x01\x02\x06\x01\x24"
#define OMP_O_MS_C_INVALID_PARAMETERS_ERROR    "\x56\x06\x01\x02\x06\x01\x25"
#define OMP_O_MS_C_ITEM                        "\x56\x06\x01\x02\x06\x01\x26"
#define OMP_O_MS_C_ITEM_TO_FORWARD             "\x56\x06\x01\x02\x06\x01\x27"
#define OMP_O_MS_C_ITEMS                       "\x56\x06\x01\x02\x06\x01\x28"
#define OMP_O_MS_C_LABEL_AND_REDIRECTION       "\x56\x06\x01\x02\x06\x01\x29"
#define OMP_O_MS_C_LABELS_AND_REDIRECTIONS     "\x56\x06\x01\x02\x06\x01\x2A"
#define OMP_O_MS_C_LIBRARY_ERROR               "\x56\x06\x01\x02\x06\x01\x2B"
#define OMP_O_MS_C_LIST_ARGUMENT               "\x56\x06\x01\x02\x06\x01\x2C"
#define OMP_O_MS_C_LIST_ATTRIBUTE_DEFAULTS     "\x56\x06\x01\x02\x06\x01\x2D"
```

```
#define OMP_O_MS_C_LIST_RESULT                "\x56\x06\x01\x02\x06\x01\x2E"
#define OMP_O_MS_C_MS_ENTRY_INFO_SELECTION    "\x56\x06\x01\x02\x06\x01\x2F"
#define OMP_O_MS_C_MS_ENTRY_INFORMATION       "\x56\x06\x01\x02\x06\x01\x30"
#define OMP_O_MS_C_ORIGINATOR_INVALID_ERROR   "\x56\x06\x01\x02\x06\x01\x31"
#define OMP_O_MS_C_PASSWORD                   "\x56\x06\x01\x02\x06\x01\x32"
#define OMP_O_MS_C_RANGE                      "\x56\x06\x01\x02\x06\x01\x33"
#define OMP_O_MS_C_RANGE_ERROR                "\x56\x06\x01\x02\x06\x01\x34"
#define OMP_O_MS_C_RECIPIENT_IMPROPER_ERROR   "\x56\x06\x01\x02\x06\x01\x35"
#define OMP_O_MS_C_REGISTER_ARGUMENT          "\x56\x06\x01\x02\x06\x01\x36"
#define OMP_O_MS_C_REGISTER_MS_ARGUMENT       "\x56\x06\x01\x02\x06\x01\x37"
#define OMP_O_MS_C_REGISTER_REJECTED_ERROR    "\x56\x06\x01\x02\x06\x01\x38"
#define OMP_O_MS_C_REMOTE_BIND_ERROR          "\x56\x06\x01\x02\x06\x01\x39"
#define OMP_O_MS_C_RESTRICTIONS               "\x56\x06\x01\x02\x06\x01\x3A"
#define OMP_O_MS_C_SECURITY_ERROR             "\x56\x06\x01\x02\x06\x01\x3B"
#define OMP_O_MS_C_SELECTOR                   "\x56\x06\x01\x02\x06\x01\x3C"
#define OMP_O_MS_C_SEQUENCE_NUMBER_ERROR      "\x56\x06\x01\x02\x06\x01\x3D"
#define OMP_O_MS_C_SEQUENCE_NUMBER_PROBLEM    "\x56\x06\x01\x02\x06\x01\x3E"
#define OMP_O_MS_C_SEQUENCE_NUMBER_RANGE      "\x56\x06\x01\x02\x06\x01\x3F"
#define OMP_O_MS_C_SERVICE_ERROR              "\x56\x06\x01\x02\x06\x01\x40"
#define OMP_O_MS_C_SESSION                    "\x56\x06\x01\x02\x06\x01\x41"
#define OMP_O_MS_C_STRONG_CREDENTIALS         "\x56\x06\x01\x02\x06\x01\x42"
#define OMP_O_MS_C_SUBMIT_CTRL_VIOLATED_ERROR "\x56\x06\x01\x02\x06\x01\x43"
#define OMP_O_MS_C_SUMMARIZE_ARGUMENT         "\x56\x06\x01\x02\x06\x01\x44"
#define OMP_O_MS_C_SUMMARY                    "\x56\x06\x01\x02\x06\x01\x45"
#define OMP_O_MS_C_SUMMARY_PRESENT            "\x56\x06\x01\x02\x06\x01\x46"
#define OMP_O_MS_C_SUMMARY_REQUESTS           "\x56\x06\x01\x02\x06\x01\x47"
#define OMP_O_MS_C_SUMMARY_RESULT             "\x56\x06\x01\x02\x06\x01\x48"
#define OMP_O_MS_C_SYSTEM_ERROR               "\x56\x06\x01\x02\x06\x01\x49"
#define OMP_O_MS_C_UNSUPT_CRITICAL_FUNC_ERROR "\x56\x06\x01\x02\x06\x01\x4A"
#define OMP_O_MS_C_WAIT_RESULT                "\x56\x06\x01\x02\x06\x01\x4B"
```

```
/* OM Attribute Names */

#define MS_ABSENT                           ((OM_type) 1201)
#define MS_Address                          ((OM_type) 1202)
#define MS_Alert_Indication                 ((OM_type) 1203)
#define MS_ALERT_ADDRESS                    ((OM_type) 1204)
#define MS_ALERT_QUALIFIER                  ((OM_type) 1205)
#define MS_ALERT_REGISTRATION_IDENTIFIER    ((OM_type) 1206)
#define MS_Allowed_Content_Types            ((OM_type) 1207)
#define MS_Allowed_EITs                     ((OM_type) 1208)
#define MS_ALTERNATE_RECIPIENT              ((OM_type) 1209)
#define MS_ALTERNATE_RECIPIENT_ALLOWED      ((OM_type) 1210)
#define MS_ATTRIBUTE_TYPE                   ((OM_type) 1211)
#define MS_ATTRIBUTE_PROBLEMS               ((OM_type) 1212)
#define MS_ATTRIBUTE_VALUE                  ((OM_type) 1213)
#define MS_ATTRIBUTES                       ((OM_type) 1214)
#define MS_Auto_Action_Deregistrations      ((OM_type) 1215)
#define MS_Auto_Action_Registrations        ((OM_type) 1216)
#define MS_Auto_Action_Request-Problems     ((OM_type) 1217)
#define MS_AUTO_FORWARD_ARGUMENTS           ((OM_type) 1218)
#define MS_AVA                              ((OM_type) 1219)
#define MS_Available_Attribute_Types        ((OM_type) 1220)
#define MS_Available_Auto_Actions           ((OM_type) 1221)
#define MS_BIND_TOKEN                       ((OM_type) 1222)
#define MS_CERTIFICATE                      ((OM_type) 1223)
#define MS_Change_Credentials               ((OM_type) 1224)
#define MS_Child_Entries                    ((OM_type) 1225)
#define MS_CONFIDENTIALITY_ALGORITHM        ((OM_type) 1226)
#define MS_CONTENT_CORRELATOR               ((OM_type) 1227)
#define MS_CONTENT_IDENTIFIER               ((OM_type) 1228)
#define MS_CONTENT_LENGTH                   ((OM_type) 1229)
#define MS_CONTENT_RETURN_REQUESTED         ((OM_type) 1230)
#define MS_CONTENT_TYPE                     ((OM_type) 1231)
#define MS_CONTENT_TYPE-INT                 ((OM_type) 1232)
#define MS_CONVERSION_LOSS_PROHIBITED       ((OM_type) 1233)
#define MS_CONVERSION_PROHIBITED            ((OM_type) 1234)
#define MS_Content_Types_Supported          ((OM_type) 1235)
#define MS_COUNT                            ((OM_type) 1236)
#define MS_CREATION_TIME_RANGE              ((OM_type) 1237)
#define MS_Default_Delivery_Controls        ((OM_type) 1238)
#define MS_DEFERRED_FELIVERY_TIME           ((OM_type) 1239)
#define MS_DELETE_AFTER_AUTO_FORWARD        ((OM_type) 1240)
#define MS_DELETE_PROBLEMS                  ((OM_type) 1241)
#define MS_Deliverable_Content_Types        ((OM_type) 1242)
#define MS_Deliverable_EIT                  ((OM_type) 1243)
#define MS_Deliverable_Max_Content_Len      ((OM_type) 1244)
#define MS_DISCLOSURE_ALLOWED               ((OM_type) 1245)
#define MS_eitS                             ((OM_type) 1246)
#define MS_ENTRY_INFORMATION                ((OM_type) 1247)
#define MS_EXPANSION_PROHIBITED             ((OM_type) 1248)
#define MS_Fetch_Attribute_Defaults         ((OM_type) 1249)
#define MS_Fetch_Restriction-Problems       ((OM_type) 1250)
#define MS_Fetch_Restrictions               ((OM_type) 1251)
#define MS_FILE_DESCRIPTOR                  ((OM_type) 1252)
#define MS_Filter                           ((OM_type) 1253)
#define MS_FROM                             ((OM_type) 1254)
#define MS_FROM-INT                         ((OM_type) 1255)
#define MS_IA5_String                       ((OM_type) 1256)
```

```
#define MS_INFORMATION_BASE_TYPE          ((OM_type) 1257)
#define MS_Initiator                      ((OM_type) 1258)
#define MS_Initiator_Credentials          ((OM_type) 1259)
#define MS_ITEM                           ((OM_type) 1260)
#define MS_ITEMS                          ((OM_type) 1261)
#define MS_LABEL_AND_REDIRECTION          ((OM_type) 1262)
#define MS_Labels_And_Redirections        ((OM_type) 1263)
#define MS_LSTEST_DELIVERY_TIME           ((OM_type) 1264)
#define MS_Limit                          ((OM_type) 1265)
#define MS_LIST                           ((OM_type) 1266)
#define MS_List_Attribute_Defaults        ((OM_type) 1267)
#define MS_Max_Content_Length             ((OM_type) 1268)
#define ms_MS_Configuration_Request       ((OM_type) 1269)
#define MS_Name                           ((OM_type) 1270)
#define MS_NEW_CREDENTIALS                ((OM_type) 1271)
#define MS_NEW_ENTRY                      ((OM_type) 1272)
#define MS_NEXT                           ((OM_type) 1273)
#define MS_OCTET_String                   ((OM_type) 1274)
#define MS_OLD_CREDENTIALS                ((OM_type) 1275)
#define MS_ORIGIN_CHECK                   ((OM_type) 1276)
#define MS_ORIGINAL_EITS                  ((OM_type) 1277)
#define MS_ORIGINATOR_CERTIFICATE         ((OM_type) 1278)
#define MS_ORIGINATOR_NAME                ((OM_type) 1279)
#define MS_ORIGINATOR_RETURN_ADDRESS      ((OM_type) 1280)
#define MS_OTHER_PARAMETERS               ((OM_type) 1281)
#define MS_Override                       ((OM_type) 1282)
#define MS_Permissible_Content_Types      ((OM_type) 1283)
#defing MS_PERMISSIBLE_CONTENT_TYPES_I    ((OM_type) 1284)
#define MS_Permissible_EITs               ((OM_type) 1285)
#define MS_Permissible_Lowest_Priority    ((OM_type) 1286)
#define MS_Permissible_Max_Content_Len    ((OM_type) 1287)
#define MS_Permissible_Operations         ((OM_type) 1288)
#define MS_PRECISE                        ((OM_type) 1289)
#define MS_PRESENT                        ((OM_type) 1290)
#define MS_PRIORITY                       ((OM_type) 1291)
#define MS_PROBLEM                        ((OM_type) 1292)
#define MS_PROOF_OF_SUBMISSION_REQUESTED  ((OM_type) 1293)
#define MS_Range                          ((OM_type) 1294)
#define MS_REASSIGNMENT_PROHIBITED        ((OM_type) 1295)
#define MS_RECIPIENTS                     ((OM_type) 1296)
#define MS_RECIPIENT_DESCRIPTORS          ((OM_type) 1297)
#define MS_REGISTRATION_ID                ((OM_type) 1298)
#define MS_REGISTRATION_PARAMETER         ((OM_type) 1299)
#define MS_REQUESTED                      ((OM_type) 1300)
#define MS_REQUESTED_ATTRIBUTES           ((OM_type) 1301)
#define MS_REQUESTED_STR                  ((OM_type) 1302)
#define MS_Responder_Credentials          ((OM_type) 1303)
#define MS_Restrict                       ((OM_type) 1304)
#define MS_Security_Context               ((OM_type) 1305)
#define MS_SECURITY_LABEL                 ((OM_type) 1306)
#define MS_SECURITY_PROBLEM               ((OM_type) 1307)
#define MS_SELECTION                      ((OM_type) 1308)
#define MS_SELECTOR                       ((OM_type) 1309)
#define MS_SEQUENCE_NUMBER                ((OM_type) 1310)
#define MS_SEQUENCE_NUMBER-PROBLEMS       ((OM_type) 1311)
#define MS_SEQUENCE_NUMBER_RANGE          ((OM_type) 1312)
#define MS_SIMPLE                         ((OM_type) 1313)
#define MS_SPAN                           ((OM_type) 1314)
```

```
#define MS_STRONG                          ((OM_type) 1315)
#define MS_SUMMARIES                       ((OM_type) 1316)
#define MS_SUMMARY_REQUESTS                ((OM_type) 1317)
#define MS_TO                              ((OM_type) 1318)
#define MS_TO-INT                          ((OM_type) 1319)
#define MS_TYPE                            ((OM_type) 1320)
#define MS_User_Security_Labels            ((OM_type) 1321)
#define MS_WAIT_NEW_AVAILABLE              ((OM_type) 1322)


/* Enumeration */
/*
 * The following enumeration tags and enumeration constants
 * are defined for use as values of the corresponding OM attributes.
 */

/* Enumeration Tags for MS_Problem: */

#define MS_E_action_type_not_subscribed       ((OM_enumeration) 1)
#define MS_E_asynchrony_not_supported         ((OM_enumeration) 2)
#define MS_E_attrib_type_not_subscribed       ((OM_enumeration) 3)
#define MS_E_authentication_error             ((OM_enumeration) 4)
#define MS_E_bad_argument                     ((OM_enumeration) 5)
#define MS_E_bad_class                        ((OM_enumeration) 6)
#define MS_E_bad_session                      ((OM_enumeration) 7)
#define MS_E_busy                             ((OM_enumeration) 8)
#define MS_E_child_entry_specified            ((OM_enumeration) 9)
#define MS_E_communications_problem           ((OM_enumeration) 10)
#define MS_E_content_length_problem           ((OM_enumeration) 11)
#define MS_E_content_type_problem             ((OM_enumeration) 12)
#define MS_E_defer_deliv_cancel_reject        ((OM_enumeration) 13)
#define MS_E_delETE_rESTRICTION_PROBLEM       ((OM_enumeration) 14)
#define MS_E_eit_problem                      ((OM_enumeration) 15)
#define MS_E_elmt_serv_not_subscribed         ((OM_enumeration) 16)
#define MS_E_feature_not_negotiated           ((OM_enumeration) 17)
#define MS_E_feature_unavailable              ((OM_enumeration) 18)
#define MS_E_inappropriate_for_operatn        ((OM_enumeration) 19)
#define MS_E_inappropriate_matching           ((OM_enumeration) 20)
#define MS_E_inconsistent_request             ((OM_enumeration) 21)
#define MS_E_invalid_attribute_value          ((OM_enumeration) 22)
#define MS_E_invalid_FEATURE                  ((OM_enumeration) 23)
#define MS_E_invalid_parameters               ((OM_enumeration) 24)
#define MS_E_message_submiT_id_invalid        ((OM_enumeration) 25)
#define MS_E_miscellaneous                    ((OM_enumeration) 26)
#define MS_E_no_such_class                    ((OM_enumeration) 27)
#define MS_E_no_such_entry                    ((OM_enumeration) 28)
#define MS_E_originator_invalid               ((OM_enumeration) 29)
#define MS_E_out_of_memory                    ((OM_enumeration) 30)
#define MS_E_recipient_improperly_specified   ((OM_enumeration) 31)
#define MS_E_register_rejected                ((OM_enumeration) 32)
#define MS_E_remote_bind_error                ((OM_enumeration) 33)
#define MS_E_reversed                         ((OM_enumeration) 34)
#define MS_E_security                         ((OM_enumeration) 35)
#define MS_E_submission_ctrl_violated         ((OM_enumeration) 36)
#define MS_E_too_many_operations              ((OM_enumeration) 37)
#define MS_E_too_many_sessions                ((OM_enumeration) 38)
#define MS_E_unable_establish_associatn       ((OM_enumeration) 39)
#define MS_E_unacceptable_securE_contxt       ((OM_enumeration) 40)
```

```
#define MS_E_unavailable                    ((OM_enumeration) 41)
#define MS_E_unavailable_action_type        ((OM_enumeration) 42)
#define MS_E_unavailable_attribute_type     ((OM_enumeration) 43)
#define MS_E_unsupported_critical_func      ((OM_enumeration) 44)
#define MS_E_unwilling_to_perfoRM           ((OM_enumeration) 45)


/* Constants */

#define MS_DEFAULT_FEATURE_LIST             ((MS_feature) 0)
#define MS_SUCCESS                          ((MS_status) 0)
#define MS_NO_WORKSPACE                     ((MS_status) 1)

/* Constants of type OM_object */

#define MS_NO_FILTER                        ((OM_object) 0)
#define MS_NO_NEW_ENTRIES                   ((OM_object) 0)
#define MS_NULL_RESULT                      ((OM_object) 0)
#define MS_OPERATION_NOT_STARTED            ((OM_object) 0)

/* Constants of type Integer */

/* Completion-Flag                          (Unsigned-Integer): */

#define MS_COMPLETED_OPERATION              ((OM_uint) 1)
#define MS_OUTSTANDING_OPERATION            ((OM_uint) 2)
#define MS_NO_OUTSTANDING_OPERATION         ((OM_uint) 3)

/* Information-Base-Type                    (Integer): */

#define MS_STORED_MESSAGES                  ((OM_integer) 0)
#define MS_INLOG                            ((OM_integer) 1)
#define MS_OUTLOG                           ((OM_integer) 2)

#endif /* XMS_HEADER */
```

9.2 <xmsga.h>

The **<xmsga.h>** header defines *MS_GENERAL_ATTRIBUTES_PACKAGE*, the constant value for the MS General Attributes Package. It also defines the object identifiers of the MS General Attribute types supported by this package (see Chapter 7).

All client programs that include this header must first include the Object Management header **<xom.h>** (see reference **XOM**) and the **<xms.h>** header.

```
/*
 *  xmsga.h   (Message Store General Attributes Package)
 */

#ifndef XMSGA_HEADER
#define XMSGA_HEADER


/* MS General Attributes Package object identifier */

#define OMP_O_MS_GENERAL_ATTRIBUTES_PACKAGE   "\x56\x06\x01\x02\x06\x02"


/* MS General Attributes Types */
/*
 * Note: Every client program must explicitly import into
 * every compilation unit (C source program) the classes or
 * Object Identifiers that it uses. Each of these classes or
 * Object Identifier names must then be explicitly exported from
 * just one compilation unit.
 * Importing and exporting can be done using the OM_IMPORT and
 * OM_EXPORT macros respectively (see the XOM API Specification).
 * For instance, the client program uses
 *                OM_IMPORT( MS_A_CHILD_SEQUENCE_NUMBERS  )
 * which in turn will make use of
 *                OMP_O_MS_A_CHILD_SEQUENCE_NUMBERS
 * defined below.
 */

#define OMP_O_MS_A_CHILD_SEQUENCE_NUMBERS       "\x56\x04\x03\x00"
#define OMP_O_MS_A_CONTENT                      "\x56\x04\x03\x01"
#define OMP_O_MS_A_CONTENT_CONFIDENTL_ALGM_ID   "\x56\x04\x03\x02"
#define OMP_O_MS_A_CONTENT_CORRELATOR           "\x56\x04\x03\x03"
#define OMP_O_MS_A_CONTENT_IDENTIFIER           "\x56\x04\x03\x04"
#define OMP_O_MS_A_CONTENT_INTEGRITY_CHECK      "\x56\x04\x03\x05"
#define OMP_O_MS_A_CONTENT_LENGTH               "\x56\x04\x03\x06"
#define OMP_O_MS_A_CONTENT_RETURNED             "\x56\x04\x03\x07"
#define OMP_O_MS_A_CONTENT_TYPE                 "\x56\x04\x03\x08"
#define OMP_O_MS_A_CONVERSION_LOSS_PROHIBITED   "\x56\x04\x03\x09"
#define OMP_O_MS_A_CONVERTED_EITS               "\x56\x04\x03\x0A"
#define OMP_O_MS_A_CREATION_TIME                "\x56\x04\x03\x0B"
#define OMP_O_MS_A_DELIVERED_EITS               "\x56\x04\x03\x0C"
#define OMP_O_MS_A_DELIVERY_FLAGS               "\x56\x04\x03\x0D"
#define OMP_O_MS_A_DL_EXPANSION_HISTORY         "\x56\x04\x03\x0E"
#define OMP_O_MS_A_ENTRY_STATUS                 "\x56\x04\x03\x0F"
#define OMP_O_MS_A_ENTRY_TYPE                   "\x56\x04\x03\x10"
#define OMP_O_MS_A_INTENDED_RECIPIENT_NAME      "\x56\x04\x03\x11"
#define OMP_O_MS_A_MESSAGE_DELIVERY_ENVELOPE    "\x56\x04\x03\x12"
#define OMP_O_MS_A_MESSAGE_DELIVERY_ID          "\x56\x04\x03\x13"
```

```
#define OMP_O_MS_A_MESSAGE_DELIVERY_TIME        "\x56\x04\x03\x14"
#define OMP_O_MS_A_MESSAGE_ORIGIN_AUTHEN_CHK    "\x56\x04\x03\x15"
#define OMP_O_MS_A_MESSAGE_SECURITY_LABEL       "\x56\x04\x03\x16"
#define OMP_O_MS_A_MESSAGE_SUBMISSION_TIME      "\x56\x04\x03\x17"
#define OMP_O_MS_A_MESSAGE_TOKEN                "\x56\x04\x03\x18"
#define OMP_O_MS_A_ORIGINAL_EITS                "\x56\x04\x03\x19"
#define OMP_O_MS_A_ORIGINATOR_CERTIFICATE       "\x56\x04\x03\x1A"
#define OMP_O_MS_A_ORIGINATOR_NAME              "\x56\x04\x03\x1B"
#define OMP_O_MS_A_OTHER_RECIPIENT_NAMES        "\x56\x04\x03\x1C"
#define OMP_O_MS_A_PARENT_SEQUENCE_NUMBER       "\x56\x04\x03\x1D"
#define OMP_O_MS_A_PERRECIP_REPORT_DELIV_FLDS   "\x56\x04\x03\x1E"
#define OMP_O_MS_A_PRIORITY                     "\x56\x04\x03\x1F"
#define OMP_O_MS_A_PROOF_OF_DELIVERY_REQUEST    "\x56\x04\x03\x20"
#define OMP_O_MS_A_REDIRECTION_HISTORY          "\x56\x04\x03\x21"
#define OMP_O_MS_A_REPORT_DELIVERY_ENVELOPE     "\x56\x04\x03\x22"
#define OMP_O_MS_A_REPORT_ORIGIN_AUTHEN_CHK     "\x56\x04\x03\x23"
#define OMP_O_MS_A_REPORTING_DL_NAME            "\x56\x04\x03\x24"
#define OMP_O_MS_A_REPORTING_MTA_CERTIFICATE    "\x56\x04\x03\x25"
#define OMP_O_MS_A_SECURITY_CLASSIFICATION      "\x56\x04\x03\x26"
#define OMP_O_MS_A_SEQUENCE_NUMBER              "\x56\x04\x03\x27"
#define OMP_O_MS_A_SUBJECT_SUBMISSION_ID        "\x56\x04\x03\x28"
#define OMP_O_MS_A_THIS_RECIPIENT_NAME          "\x56\x04\x03\x29"
```

```
/* Enumeration Constants */

/* for MS_A_ENTRY_STATUS */

#define MS_ES_NEW                    ((OM_enumeration) 0)
#define MS_ES_LISTED                 ((OM_enumeration) 1)
#define MS_ES_PROCESSED              ((OM_enumeration) 2)


/* for MS_A_ENTRY_TYPE */

#define MS_ET_DELIVERED_MESSAGE      ((OM_enumeration) 0)
#define MS_ET_DELIVERED_REPORT       ((OM_enumeration) 1)
#define MS_ET_RETURNED_CONTENT       ((OM_enumeration) 2)


/* for MS_A_PRIORITY */

#define MS_PTY_NORMAL                ((OM_enumeration) 0)
#define MS_PTY_LOW                   ((OM_enumeration) 1)
#define MS_PTY_URGENT                ((OM_enumeration) 2)


/* for MS_A_SECURITY_CLASSIFICATION */

#define MS_SC_UNMARKED               ((OM_enumeration) 0)
#define MS_SC_UNCLASSIFIED           ((OM_enumeration) 1)
#define MS_SC_RESTRICTED             ((OM_enumeration) 2)
#define MS_SC_CONFIDENTIAL           ((OM_enumeration) 3)
#define MS_SC_SECRET                 ((OM_enumeration) 4)
#define MS_SC_TOP_SECRET             ((OM_enumeration) 5)


#endif /* XMSGA_HEADER */
```

9.3 <xmsima.h>

The **<xmsima.h>** header defines *MS_IM_ATTRIBUTES_PACKAGE*, the constant value for the MS Interpersonal Messaging Package. It also defines the object identifiers of the MS Interpersonal Messaging Attribute types supported by this package (see Chapter 8).

All client programs that include this header must first include the Object Management header **<xom.h>** (see reference **XPOM**) and the **<xms.h>** header.

```
/*
 *  xmsima.h   (Message Store Interpersonal Messaging Attributes Package)
 */

#ifndef XMSIMA_HEADER
#define XMSIMA_HEADER


/* MS Interpersonal Messaging Attributes Package object identifier */

#define OMP_O_MS_IM_ATTRIBUTES_PACKAGE    "\x56\x06\x01\x02\x06\x03"


/* MS Interpersonal Messaging Attributes Types */

/*
 * Note: Every client program must explicitly import into
 * every compilation unit (C source program) the classes or
 * Object Identifiers that it uses. Each of these classes or
 * Object Identifier names must then be explicitly exported from
 * just one compilation unit.
 * Importing and exporting can be done using the OM_IMPORT and
 * OM_EXPORT macros respectively (see the XOM API Specification).
 * For instance, the client program uses
 *                OM_IMPORT( MS_IM_ACKNOWLEDGMENT_MODE  )
 * which in turn will make use of
 *                OMP_O_MS_IM_ACKNOWLEDGMENT_MODE
 * defined below.
 */

#define OMP_O_MS_IM_A_ACKNOWLEDGMENT_MODE        "\x56\x01\x09\x09"
#define OMP_O_MS_IM_A_AUTHORIZING_USERS          "\x56\x01\x07\x0A"
#define OMP_O_MS_IM_A_AUTO_FORWARD_COMMENT       "\x56\x01\x09\x06"
#define OMP_O_MS_IM_A_AUTO_FORWARDED             "\x56\x01\x07\x09"
#define OMP_O_MS_IM_A_BILATERAL_DEF_BODY_PARTS   "\x56\x01\x08\x0A"
#define OMP_O_MS_IM_A_BLIND_COPY_RECIPIENTS      "\x56\x01\x07\x0D"
#define OMP_O_MS_IM_A_BODY                       "\x56\x01\x08\x00"
#define OMP_O_MS_IM_A_CONVERSION_EITS            "\x56\x01\x09\x03"
#define OMP_O_MS_IM_A_COPY_RECIPIENTS            "\x56\x01\x07\x0C"
#define OMP_O_MS_IM_A_DISCARD_REASON             "\x56\x01\x09\x05"
#define OMP_O_MS_IM_A_EXPIRY_TIME                "\x56\x01\x07\x05"
#define OMP_O_MS_IM_A_EXTENDED_BODY_PART_TYPES   "\x56\x01\x08\x0C"
#define OMP_O_MS_IM_A_G3_FAX_BODY_PARTS          "\x56\x01\x08\x03"
#define OMP_O_MS_IM_A_G3_FAX_DATA                "\x56\x01\x08\x16"
#define OMP_O_MS_IM_A_G3_FAX_PARAMETERS          "\x56\x01\x08\x0F"
#define OMP_O_MS_IM_A_G4_CLASS1_BODY_PARTS       "\x56\x01\x08\x04"
#define OMP_O_MS_IM_A_HEADING                    "\x56\x01\x07\x00"
#define OMP_O_MS_IM_A_IA5_TEXT_BODY_PARTS        "\x56\x01\x08\x01"
#define OMP_O_MS_IM_A_IA5_TEXT_DATA              "\x56\x01\x08\x14"
```

```
#define OMP_O_MS_IM_A_IA5_TEXT_PARAMETERS        "\x56\x01\x08\x0D"
#define OMP_O_MS_IM_A_IMPORTANCE                 "\x56\x01\x07\x07"
#define OMP_O_MS_IM_A_INCOMPLETE_COPY            "\x56\x01\x07\x11"
#define OMP_O_MS_IM_A_IPM_ENTRY_TYPE             "\x56\x01\x06\x00"
#define OMP_O_MS_IM_A_IPM_PREFERRED_RECIPIENT    "\x56\x01\x09\x02"
#define OMP_O_MS_IM_A_IPM_SYNOPSIS               "\x56\x01\x06\x01"
#define OMP_O_MS_IM_A_IPN_ORIGINATOR             "\x56\x01\x09\x01"
#define OMP_O_MS_IM_A_LANGUAGES                  "\x56\x01\x07\x12"
#define OMP_O_MS_IM_A_MESSAGE_BODY_PARTS         "\x56\x01\x08\x08"
#define OMP_O_MS_IM_A_MESSAGE_DATA               "\x56\x01\x08\x1A"
#define OMP_O_MS_IM_A_MESSAGE_PARAMETERS         "\x56\x01\x08\x13"
#define OMP_O_MS_IM_A_MIXED_MODE_BODY_PARTS      "\x56\x01\x08\x09"
#define OMP_O_MS_IM_A_NATIONAL_DEF_BODY_PARTS    "\x56\x01\x08\x0B"
#define OMP_O_MS_IM_A_NON_RECEIPT_REASON         "\x56\x01\x09\x04"
#define OMP_O_MS_IM_A_NRN_REQUESTORS             "\x56\x01\x07\x14"
#define OMP_O_MS_IM_A_OBSOLETED_IPMS             "\x56\x01\x07\x0E"
#define OMP_O_MS_IM_A_ORIGINATOR                 "\x56\x01\x07\x02"
#define OMP_O_MS_IM_A_PRIMARY_RECIPIENTS         "\x56\x01\x07\x0B"
#define OMP_O_MS_IM_A_RECEIPT_TIME               "\x56\x01\x09\x08"
#define OMP_O_MS_IM_A_RELATED_IPMS               "\x56\x01\x07\x0F"
#define OMP_O_MS_IM_A_REPLIED_TO_IPM             "\x56\x01\x07\x03"
#define OMP_O_MS_IM_A_REPLY_RECIPIENTS           "\x56\x01\x07\x10"
#define OMP_O_MS_IM_A_REPLY_REQUESTORS           "\x56\x01\x07\x15"
#define OMP_O_MS_IM_A_REPLY_TIME                 "\x56\x01\x07\x06"
#define OMP_O_MS_IM_A_RETURNED_IPM               "\x56\x01\x09\x07"
#define OMP_O_MS_IM_A_RN_REQUESTORS              "\x56\x01\x07\x13"
#define OMP_O_MS_IM_A_SENSITIVITY                "\x56\x01\x07\x08"
#define OMP_O_MS_IM_A_SUBJECT                    "\x56\x01\x07\x04"
#define OMP_O_MS_IM_A_SUBJECT_IPM                "\x56\x01\x09\x00"
#define OMP_O_MS_IM_A_SUPPL_RECEIPT_INFO         "\x56\x01\x09\x0A"
#define OMP_O_MS_IM_A_TELETEX_BODY_PARTS         "\x56\x01\x08\x05"
#define OMP_O_MS_IM_A_TELETEX_DATA               "\x56\x01\x08\x17"
#define OMP_O_MS_IM_A_TELETEX_PARAMETERS         "\x56\x01\x08\x10"
#define OMP_O_MS_IM_A_THIS_IPM                   "\x56\x01\x07\x01"
#define OMP_O_MS_IM_A_VIDEOTEX_BODY_PARTS        "\x56\x01\x08\x06"
#define OMP_O_MS_IM_A_VIDEOTEX_DATA              "\x56\x01\x08\x18"
#define OMP_O_MS_IM_A_VIDEOTEX_PARAMETERS        "\x56\x01\x08\x11"
```

```
/* Classes in the MS IM Attributes Package */

#define OMP_O_MS_IM_C_BODY                          "\x56\x06\x01\x02\x06\x03\x01"
#define OMP_O_MS_IM_C_G3_FAX_DATA                   "\x56\x06\x01\x02\x06\x03\x02"
#define OMP_O_MS_IM_C_BODYPART_SYNOPSIS             "\x56\x06\x01\x02\x06\x03\x03"
#define OMP_O_MS_IM_C_HEADING                       "\x56\x06\x01\x02\x06\x03\x04"
#define OMP_O_MS_IM_C_IPM_SYNOPSIS                  "\x56\x06\x01\x02\x06\x03\x05"
#define OMP_O_MS_IM_C_MSG_BODYPART_SYNOPSIS         "\x56\x06\x01\x02\x06\x03\x06"
#define OMP_O_MS_IM_C_NON_MSG_BODYPART_SYNOPSIS     "\x56\x06\x01\x02\x06\x03\x07"
#define OMP_O_MS_IM_C_TELETEX_DATA                  "\x56\x06\x01\x02\x06\x03\x08"
#define OMP_O_MS_IM_C_TELETEX_PARAMETERS            "\x56\x06\x01\x02\x06\x03\x09"



/* OM Attribute Names in the MS IM Attributes Package */

#define MS_IM_AUTHORIZING_USERS                  ((OM_type) 1401)
#define MS_IM_AUTO_FORWARDED                     ((OM_type) 1402)
#define MS_IM_BLIND_COPY_RECIPIENTS              ((OM_type) 1403)
#define MS_IM_BODY_PART                          ((OM_type) 1404)
#define MS_IM_BODY_PART_SYNOPSIS                 ((OM_type) 1405)
#define MS_IM_COPY_RECIPIENTS                    ((OM_type) 1406)
#define MS_IM_EXPIRY_TIME                        ((OM_type) 1407)
#define MS_IM_EXTENSIONS                         ((OM_type) 1408)
#define MS_IM_IMAGES                             ((OM_type) 1409)
#define MS_IM_IMPORTANCE                         ((OM_type) 1410)
#define MS_IM_MESSAGE_BODY_PART_SYNOPSIS         ((OM_type) 1411)
#define MS_IM_NON_MESSAGE_BODY_PART_SYNOPSIS     ((OM_type) 1412)
#define MS_IM_NUMBER_OF_PAGES                    ((OM_type) 1413)
#define MS_IM_OBSOLETED_IPMS                     ((OM_type) 1414)
#define MS_IM_ORIGINATOR                         ((OM_type) 1415)
#define MS_IM_PARAMETERS                         ((OM_type) 1416)
#define MS_IM_PRIMARY_RECIPIENTS                 ((OM_type) 1417)
#define MS_IM_PROCESSED                          ((OM_type) 1418)
#define MS_IM_RELATED_IPMS                       ((OM_type) 1419)
#define MS_IM_REPLIED_TO_IPM                     ((OM_type) 1420)
#define MS_IM_REPLY_RECIPIENTS                   ((OM_type) 1421)
#define MS_IM_REPLY_TIME                         ((OM_type) 1422)
#define MS_IM_SENSITIVITY                        ((OM_type) 1423)
#define MS_IM_SEQUENCE_NUMBER                    ((OM_type) 1424)
#define MS_IM_SIZE                               ((OM_type) 1425)
#define MS_IM_SUBJECT                            ((OM_type) 1426)
#define MS_IM_SYNOPSIS                           ((OM_type) 1427)
#define MS_IM_TELETEX_DOCUMENT                   ((OM_type) 1428)
#define MS_IM_THIS_IPM                           ((OM_type) 1429)
#define MS_IM_TYPE                               ((OM_type) 1430)
```

```
/* Enumeration Constants */

/* for MS_IM_ACKNOWLEDGEMENT_MODE  */
#define MS_AM_AUTOMATIC                 ((OM_enumeration) 0)
#define MS_AM_MANUAL                    ((OM_enumeration) 1)

/* for MS_IM_DISCARD_REASON  */
#define MS_DR_NO_DISCARD                ((OM_enumeration) -1)
#define MS_DR_IPM_EXPIRED               ((OM_enumeration) 0)
#define MS_DR_IPM_OBSOLETED             ((OM_enumeration) 1)
#define MS_DR_USER_TERMINATED           ((OM_enumeration) 2)

/* MS_IM_IA5_REPERTOIRE  */
#define MS_IR_IA5                       ((OM_enumeration) 2)
#define MS_IR_ITA2                      ((OM_enumeration) 5)

/* MS_IM_IMPORTANCE  */
#define MS_IM_LOW                       ((OM_enumeration) 0)
#define MS_IM_ROUTINE                   ((OM_enumeration) 1)
#define MS_IM_HIGH                      ((OM_enumeration) 2)

/* MS_IM_IPM_ENTRY_TYPE  */
#define MS_IE_IPM                       ((OM_enumeration) 0)
#define MS_IE_RN                        ((OM_enumeration) 1)
#define MS_IE_NRN                       ((OM_enumeration) 2)

/* MS_IM_NR_REASON  */
#define MS_NR_IPM_AUTO_FORWARDED        ((OM_enumeration) 0)
#define MS_NR_IPM_DISCARDED             ((OM_enumeration) 1)

/* MS_IM_SENSITIVITY  */
#define MS_SE_NOT_SENSITIVE             ((OM_enumeration) 0)
#define MS_SE_PERSONAL                  ((OM_enumeration) 1)
#define MS_SE_PRIVATE                   ((OM_enumeration) 2)
#define MS_SE_COMPANY_CONFIDENTIAL      ((OM_enumeration) 3)

/* MS_IM_VIDEOTEX_SYNTAX  */
#define MS_VS_UNSPECIFIED               ((OM_enumeration) -1)
#define MS_VS_IDS                       ((OM_enumeration) 0)
#define MS_VS_DATA_SYNTAX_1             ((OM_enumeration) 1)
#define MS_VS_DATA_SYNTAX_2             ((OM_enumeration) 2)
#define MS_VS_DATA_SYNTAX_3             ((OM_enumeration) 3)

#endif /* XMSIMA_HEADER */
```

Chapter 10

A Programming Example

This chapter provides an example of a client program, written in C, that uses this interface to the Message Store. It uses the synchronous mode. Note that this sample program is presented for illustrative purposes only and is not complete.

```
/*
 * ------------------------------------------------------------------
 *
 *        Sample client program that uses the X.400 MS interface
 *        for listing ‘‘new’’ entries from a Message Store
 *        in the synchronous mode.
 *
 *        The main objective of this program is to show how to
 *         - retrieve selected attributes of ‘‘new’’ messages.
 *
 *        Warning: This sample program is purely illustrative and
 *                 is not complete.
 *
 * ------------------------------------------------------------------
 */

/*
 * Include the relevant header files.
 */
#include   <xom.h>     /* Object Management header */
#include   <xms.h>     /* MS Package header */
#include   <xmsga.h>   /* MS General Attributes Package header */
#include   <xds.h>     /* for building OR-Name for ‘‘initiator’’
                          in Bind-Argument */
/*
 * Assume the availability of macros, e.g., CHECK_MS_CALL,
 * CHECK_OM_CALL in a header file, example.h, which will
 * define simple Error Handling Modules that is similar
 * to that for the programming example in the XDS API Specification.
 *
 * These macros check if the function returns with an error,
 * in which case an error message is logged and the program exits.
 */
#include      "example.h"

/*
 * Define the necessary Object Identifier constants that identify
 * the OM classes used in this example. (See the XOM API
 * Specification.)
 */
OM_EXPORT( DS_C_AVA )
OM_EXPORT( DS_C_DS_RDN )
OM_EXPORT( DS_C_DS_DN )
OM_EXPORT( DS_A_COUNTRY_NAME )
OM_EXPORT( DS_A_ORGANIZATION_NAME )
OM_EXPORT( DS_A_ORGANIZATIONAL_UNIT_NAME )
OM_EXPORT( DS_A_COMMON_NAME )
OM_EXPORT( MS_C_BIND_ARGUMENT )
OM_EXPORT( MS_C_FILTER_ITEMS )
OM_EXPORT( MS_C_FILTER )
OM_EXPORT( MS_C_SELECTOR )
OM_EXPORT( MS_C_ATTRIBUTE_SELECTION )
OM_EXPORT( MS_C_MS_ENTRY_INFO_SELECTION)
```

```
OM_EXPORT( MS_C_LIST_ARGUMENT )
OM_EXPORT( MS_A_ENTRY_STATUS )
OM_EXPORT( MS_A_CONTENT_LENGTH )
OM_EXPORT( MS_A_ORIGINATOR_NAME )

int main(void)
{
/*
 * Declarations
 */
OM_return_code      return_code,    /* for OM functions */
                    error;          /* for MS functions */

/*
 * ------------------------------------------------------------------------
 * For ms_initialize( ):
 *        Note: the service implicitly provides the mandatory features
 *        (i.e., MS_PACKAGE, MS_GENERAL_ATTRIBUTES_PACKAGE, MS_FU)
 *        and for this example, no additional features are being
 *        requested.
 */
static MS_feature    feature_list[] = {
                            { {0, NULL}, OM_TRUE },
                     };
OM_sint              max_sessions;
OM_sint              max_outstanding_operations;
OM_workspace         workspace;

/*
 * ------------------------------------------------------------------------
 * For ms_bind( ):
 *
 *        Build the Bind-Argument with attributes:
 *                initiator, initiator_credentials,
 *                ms_configuration_request
 */
/*
 *        Build OR-Name for the ‘‘initiator’’.
 */
static OM_descriptor                country[] {
  OM_OID_DESC(OM_CLASS, DS_C_AVA),
  OM_OID_DESC(DS_ATTRIBUTE_TYPE, DS_A_COUNTRY_NAME),
  {DS_ATTRIBUTE_VALUES, OM_S_PRINTABLE_STRING, OM_STRING("us")},
  OM_NULL_DESCRIPTOR
};
static OM_descriptor                organization[] {
  OM_OID_DESC(OM_CLASS, DS_C_AVA),
  OM_OID_DESC(DS_ATTRIBUTE_TYPE, DS_A_ORGANIZATION_NAME),
  {DS_ATTRIBUTE_VALUES, OM_S_PRINTABLE_STRING, OM_STRING("ABC Inc")},
  OM_NULL_DESCRIPTOR
};
static OM_descriptor                organization_unit[] {
  OM_OID_DESC(OM_CLASS, DS_C_AVA),
  OM_OID_DESC(DS_ATTRIBUTE_TYPE, DS_A_ORGANIZATIONAL_UNIT_NAME),
  {DS_ATTRIBUTE_VALUES, OM_S_PRINTABLE_STRING,
                                      OM_STRING("Music")},
  OM_NULL_DESCRIPTOR
};
```

```
static OM_descriptor                common_name[] {
  OM_OID_DESC(OM_CLASS,  DS_C_AVA),
  OM_OID_DESC(DS_ATTRIBUTE_TYPE,  DS_A_COMMON_NAME),
  {DS_ATTRIBUTE_VALUES,  OM_S_PRINTABLE_STRING,  OM_STRING("Ian  Doe")},
  OM_NULL_DESCRIPTOR
};
static OM_descriptor                rdn1[] = {
  OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
  {DS_AVAS,  OM_S_OBJECT,  {0, country} },
  OM_NULL_DESCRIPTOR
};
static OM_descriptor                rdn2[] = {
  OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
  {DS_AVAS,  OM_S_OBJECT,  {0, organization} },
  OM_NULL_DESCRIPTOR
};
static OM_descriptor                rdn3[] = {
  OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
  {DS_AVAS,  OM_S_OBJECT,  {0, organizational_unit} },
  OM_NULL_DESCRIPTOR
};
static OM_descriptor                rdn4[] = {
  OM_OID_DESC(OM_CLASS,  DS_C_DS_RDN),
  {DS_AVAS,  OM_S_OBJECT,  {0, common_name} },
  OM_NULL_DESCRIPTOR
};

OM_descriptor                       initiator[] = {
  OM_OID_DESC(OM_CLASS,  DS_C_DS_DN),
  {DS_RDNS,  OM_S_OBJECT,  {0, rdn1} },
  {DS_RDNS,  OM_S_OBJECT,  {0, rdn2} },
  {DS_RDNS,  OM_S_OBJECT,  {0, rdn3} },
  {DS_RDNS,  OM_S_OBJECT,  {0, rdn4} },
  OM_NULL_DESCRIPTOR
};

/*
 * In a similar manner, insert code to build
 * the ‘‘initiator_credentials’ ’ OM attribute for ‘‘bind_argument’ ’.
 *
 * Then, combine all OM attributes to build ‘‘bind_argument’ ’.
 */
OM_descriptor                        bind_argument[] = {
        OM_OID_DESC(OM_CLASS,  MS_C_BIND_ARGUMENT),
        {MS_INITIATOR,  OM_S_OBJECT,  {0, initiator} },
        {MS_INITIATOR_CREDENTIALS,  OM_S_OBJECT,
                                        {0, initiator_credentials}  },
        {MS_MS_CONFIGURATION_REQUEST,  OM_S_BOOLEAN,
                                        {OM_FALSE,  NULL} },
        OM_NULL_DESCRIPTOR
};
OM_private_object           bind_result;
OM_private_object           session;


/*
 * ------------------------------------------------------------------------
 * For ms_list( ):
```

```
 *         List-Argument:
 *                selector,
 *                requested-attributes
 */
/*
 * List-Argument: selector:
 *        Build ‘‘selector’’ such that there is a filter for matching
 *        MS attribute, Entry-Status, with value ‘‘new’’.
 */
static OM_descriptor        filter_items[] = {
  OM_OID_DESC(OM_CLASS, MS_C_FILTER_ITEMS),
  {DS_ATTRIBUTE_TYPE, OM_S_OBJECT_IDENTIFIER_STRING,
                OM_OID_DESC(OM_CLASS, MS_A_ENTRY_STATUS) },
  {DS_ATTRIBUTE_VALUES, OM_S_ENUMERATION, {MS_ES_NEW, NULL}},

  {DS_FILTER_ITEM_TYPE, OM_S_ENUMERATION, {DS_EQUALITY, NULL}},
  OM_NULL_DESCRIPTOR
};
static OM_descriptor        filter[] = {
  OM_OID_DESC(OM_CLASS, MS_C_FILTER),
  {DS_FILTER_ITEMS, OM_S_OBJECT, {0, filter_items}},
  {DS_FILTER_TYPE, OM_S_ENUMERATION, {DS_AND, NULL}},
  OM_NULL_DESCRIPTOR
};
static OM_descriptor        selector[] = {
  OM_OID_DESC(OM_CLASS, MS_C_SELECTOR),
  {MS_CHILD_ENTRIES, OM_S_BOOLEAN, {OM_FALSE, NULL} },
  {MS_FILTER, OM_S_OBJECT, {0, filter} },
  OM_NULL_DESCRIPTOR
};

/*
 * List-Argument: requested-attributes:
 *        Build ‘‘requested-attributes’’ so as to request that
 *        the result returned by the ms_list( ) give
 *        the MS attributes:
 *                - Content-Length and Originator-Name.
 */
static OM_descriptor        selection1[] = {
  OM_OID_DESC(OM_CLASS, MS_C_ATTRIBUTE_SELECTION),
  {MS_ATTRIBUTE_TYPE, OM_S_OBJECT_IDENTIFIER_STRING,
                OM_OID_DESC(OM_CLASS, MS_A_CONTENT_LENGTH) },
  OM_NULL_DESCRIPTOR
};
static OM_descriptor        selection2[] = {
  OM_OID_DESC(OM_CLASS, MS_C_ATTRIBUTE_SELECTION),
  {MS_ATTRIBUTE_TYPE, OM_S_OBJECT_IDENTIFIER_STRING,
                OM_OID_DESC(OM_CLASS, MS_A_ORIGINATOR_NAME) },
  OM_NULL_DESCRIPTOR
};
static OM_descriptor        requested_attr[] = {
  OM_OID_DESC(OM_CLASS, MS_C_MS_ENTRY_INFO_SELECTION),
  {MS_SELECTION, OM_S_OBJECT, {0, selection1} },
  {MS_SELECTION, OM_S_OBJECT, {0, selection2} },
  OM_NULL_DESCRIPTOR
};
```

```
/*
 * Then, combine all OM attributes to build ‘‘list_argument’’.
 */
OM_descriptor               list_argument[] = {
        OM_OID_DESC(OM_CLASS,  MS_C_LIST_ARGUMENT),
        {MS_SELECTOR,  OM_S_OBJECT,  {0, selector} },
        {MS_REQUESTED_ATTRIBUTES,  OM_S_OBJECT,
                                        {0, requested_attr} },
        OM_NULL_DESCRIPTOR
};

OM_private_object           list_result;
MS_invoke_id                invoke_id;

/*
 * Variables required for extracting the MS attributes
 * that were requested in the ms_list( ).
 */
OM_type              entry_types[] = {MS_REQUESTED,  0};
OM_descriptor        *entries;
OM_type              entryInfo_types[] =
                     {MS_SEQUENCE_NUMBERS,  MS_ATTRIBUTES,  0};
OM_descriptor        *entryInfos;
OM_type              reqAttributes_types[]  = {MS_ATTRIBUTE_VALUES,  0};
OM_descriptor        *reqAttributes;
OM_value_position    total_requested;
OM_value_position    total_number;
int                  i;


/*
 * ====================================================================
 * Start up the MS API and obtain a workspace.
 */
max_sessions                    = 1;
max_outstanding_operations      = 0;     /* synchronous mode */
CHECK_MS_CALL( ms_initialize    ( feature_list,
                                  &max_sessions,
                                  &max_outstanding_operations,
                                  &workspace ) );


/*
 * Establish a session with the MS.
 */
CHECK_MS_CALL( ms_bind( bind_argument,
                        &bind_result,
                        &session ) );

/*
 * Use ms_list( ) to
 * search for MS entries whose Entry Status is ‘‘new’’ and
 * return the Sequence-Number and Originator-Name MS attributes
 * of these entries.
 */
CHECK_MS_CALL( ms_list( session, list_argument,
                        &list_result,
                        &invoke_id ) );
```

```
/*
 * Extract the list of entries returned.
 */
CHECK_OM_CALL( om_get(list_result,
                      OM_EXCLUDE_ALL_BUT_CERTAIN_TYPES  +
                      OM_EXCLUDE_SUBOBJECTS,
                      entry_types, OM_FALSE,
                      0, 20,
                      &entries, &total_requested ) );

/*
 * For each of entry information requested returned by
 * ms_list( ), retrieve the MS attributes requested and
 * perform whatever processing needed.
 */
for (i=0, ientry=entries;  i < total_requested; i++, ientry++ )
{
        CHECK_OM_CALL( om_get(ientry->value.object.object,
                      OM_EXCLUDE_ALL_BUT_CERTAIN_TYPES  +
                      OM_EXCLUDE_SUBOBJECTS,
                      entryInfo_types, OM_FALSE, 0, 0,
                      &entryInfos, &total_number ) );

        CHECK_OM_CALL( om_get(entryInfos->value.object.object,
                      OM_EXCLUDE_ALL_BUT_CERTAIN_TYPES  +
                      OM_EXCLUDE_SUBOBJECTS,
                      reqAttributes_types, OM_FALSE, 0, 0,
                      &reqAttributes, &total_number ) );

    /*
     * Perform processing of entry information here...
     */
        display_SequenceNumber( entryInfos );
        display_ReqAttributes( reqAttributes );

} /* end for */

/*
 * Terminate the session with MS
 */
CHECK_MS_CALL( ms_unbind( session ) );

/*
 * Clean up
 */
CHECK_OM_CALL( om_delete( bind_result ) );
CHECK_OM_CALL( om_delete( list_result ) );
CHECK_OM_CALL( om_delete( entries ) );
CHECK_OM_CALL( om_delete( entryInfos ) );
CHECK_OM_CALL( om_delete( reqAttributes ) );

/*
 * Terminate MS API, releasing storage for the workspace
 */
ms_shutdown( );

} /* end main */
```

Appendix A

Runtime Binding

This appendix is not an integral part of the document. It describes how, in the context of selected operating systems, the C implementation of a client may be bound at runtime to the C implementation of the service.

A.1 OS/2

Binding of client applications to a service implementation at runtime under OS/2 is accomplished through the use of Dynamic Link Libraries (DLLs). Each interface of a service implementation should be presented to the client as a separate DLL. The service API functions for each library are specified as IMPORTS in the client application's definition (.DEF) file and declared as externals in the client code. This allows the client application to be compiled and linked in the absence of the actual service libraries. The OS/2 kernel recognises these unresolved external references in the client application when it is executed and loads the appropriate service DLL.

A.1.1 Service Provider Requirements

Each service interface should be implemented as a separate DLL. Furthermore, different vendor's service implementations must adhere to a consistent DLL naming convention so that client applications may include the service DLL name in their definition files.

Specifically, Message Store interface library is named:

msxapia.dll

Other related interfaces not specifically described by this document should use the same convention, e.g., the Directory Service interface library should be named:

dsxapia.dll

All API functions exposed to the client should use Pascal calling conventions (function parameters are pushed left to right and the called routine clears the stack). This is achieved by declaring each API function within an interface DLL to be *far pascal*. Other functions, which are used only within a particular library and not exposed to a client application, may use any calling convention.

Service providers are free to specify any parameters in the definition files associated with the interface libraries as they see fit subject to these restrictions, e.g., a service provider may decide to specify an initialisation routine to be executed when a service library is first loaded (the LIBRARY INITINSTANCE directive).

A.1.2 Client Application Requirements

The client application writer must specify a definition file for the client application program and enumerate the API function routines used by the application in the IMPORTS section of the file. The client application will also need to declare each imported API function as using Pascal calling conventions by declaring them as *external far Pascal*. An example of a definition file (client.def) used by a client application using the MS and OM APIs is given below.

```
NAME CLIENT
IMPORTS
       MSXAPIA.MS_BIND
       MSXAPIA.MS_CANCEL_SUBMISSION
       MSXAPIA.MS_CHECK_ALERT
       MSXAPIA.MS_DELETE
       MSXAPIA.MS_FETCH
       MSXAPIA.MS_INITIALIZE
       MSXAPIA.MS_LIST
       MSXAPIA.MS_RECEIVE_RESULT
       MSXAPIA.MS_REGISTER
       MSXAPIA.MS_REGISTER_MS
       MSXAPIA.MS_SHUTDOWN
       MSXAPIA.MS_SUBMIT
       MSXAPIA.MS_SUMMARIZE
       MSXAPIA.MS_UNBIND
       MSXAPIA.MS_WAIT
```

Each service DLL used by a client application must be placed in a directory specified in the LIBPATH directive in the OS/2 config.sys file.

A.2 UNIX System V Release 4.0

Runtime binding in UNIX System V Release 4.0 is accomplished through the use of dynamic shared libraries. Each API must be implemented as a dynamic shared library in the:

```
/usr/lib/XAPI
```

directory. Specifically, the MS interface functions will be placed in the:

```
/usr/lib/XAPI/libMS.so
```

library.

Each dynamic shared library must be implemented to comply with the UNIX System V Release 4.0 System V Application Binary Interface (ABI) Specification. There are no additional constraints on the implementation of these libraries. The libraries should, however, be well designed in order to avoid potential performance degradation that can be caused by the use of shared libraries. The **UNIX System V Release 4.0 Programmer's Guide: ANSI C and Programming Support Tools** provides information on how to design shared libraries.

All client applications must be compiled to use the functions in these dynamic libraries. Details on how to link dynamic shared libraries can be found in the **UNIX System V Release 4.0 Programmer's Guide: ANSI C and Programming Support Tools**. In particular, the client application that uses the APIs may be compiled using absolute path names for the API shared libraries or using the **-L/usr/lib/XAPI** option of the *cc* command.

Both service and client must be compiled with header files that include consistent binary bindings for all values passed between the client and the service. Consistent definitions of the library interface as specified in the main body of this document.

Glossary

This section is a glossary of terms used in this document. Certain terms are used to describe both the Message Store and Object Management - for example, attribute. In these cases, the terms are distinguished by (M) and (O):

(M) reference adapted from the Message Store Standards.

(O) reference adapted from the OSI-Abstract-Data Manipulation API (XOM) Specification.

Words in the explanation that are set in italics denote cross-references to other terms listed in the glossary.

Abstract Class (O)
An OM class of OM object of which instances are forbidden. An abstract class typically serves to document the similarities between instances of two or more concrete classes.

Abstract Syntax Notation One (ASN.1)
A notation which both enables complicated types to be defined and also enables values of these types to be specified. See reference **X.409** listed in **Referenced Documents**.

Alert Operation (M)
An MS abstract operation which allows the MS to signal, based on selection criteria, to the User Agent that messages or reports are waiting in the MS. This operation can only be issued on an existing session (or association) with the MS.

Argument
Information which is passed to a function and which specifies the details of the processing to be performed.

ASN.1
See *Abstract Syntax Notation One.*

Asynchronous operation
An operation that does not of itself cause the process requesting the operation to be blocked from further use of the CPU. This implies that the process and the operation are running concurrently.

Attribute (M)
The information of a particular type concerning an entry in an information base of the Message Store.

Attribute (Object) (O)
See *OM attribute.*

Attribute Syntax (O)
A definition of the set of values which an attribute may assume. It includes the data type, in ASN.1, and, usually, one or more matching rules by which values may be compared.

Attribute Type (M)
The component of an attribute which indicates the class of information given by that attribute. It is an Object Identifier and so is unique.

Attribute Type (Object) (O)
Any of various categories into which the client dynamically groups values on the basis of their semantics. It is an integer, unique only within the package.

Attribute Value Assertion (AVA) (M)
A proposition, which may be true, false or undefined, concerning the values of attributes in an entry.

Attribute Value Syntax (O)
See *Syntax(Object)*.

Attribute Value (M)
A particular instance of that class of information indicated by an attribute type.

Attribute Value (Object) (O)
An atomic information object.

Auto Action Type (M)
An auto action type is used to indicate the type of auto action, e.g., Alert.

Auto Action (M)
Actions that can be performed automatically by the MS, based on previously registered information from the MS-owner via the User Agent.

Auto Alert (M)
An auto action, within the MS, triggered by delivery of a message and which, based on the registered criteria, may cause an alert to be generated.

Auto Forward (M)
The auto action, within the MS, which triggers a message to be auto-forwarded to another recipient (or other recipients) by the MS. The message may optionally be deleted.

Basic Encoding Rules
A set of rules used to encode ASN.1 values as strings of octets.

BER
See *Basic Encoding Rules*.

Child Entry (M)
An entry, other than the main-entry in an information base. The parent-entry for a child-entry can be either the main-entry or another child-entry depending on the number of levels in each case.

Class (Object)
See *OM Class*.

Concrete Class (O)
An OM class of which instances are permitted.

Content Length (M)
An attribute which gives the length of the content of a delivered message (or returned content).

Content Returned (M)
An attribute which signals that a delivered report (or a delivered message) contained a returned content.

Converted EITs (M)
An attribute identifying the encoded information types of the message content after

conversion.

Creation Time (M)
An attribute which gives the creation time (by the MS) of an entry.

Delete Operation (M)
An MS abstract operation used to delete one or more entries from an information base.

Delivered EITs (M)
A multi-valued attribute that gives information about EITs in a delivered message.

Delivered Message Entry (M)
An entry in the stored messages information base resulting from a delivered message.

Delivered Report Entry (M)
An entry in the stored messages information base resulting from a delivered report.

Descriptor (O)
A defined data structure which is used to represent an OM attribute type and a single value.

Descriptor List (O)
An ordered sequence of descriptors which is used to represent several OM attribute types and values.

Directory
A collection of open systems which cooperate to hold a logical database of information about a set of objects in the real world.

Entry (M)
An information set in an information base. See main-entry, parent-entry and child-entry for further classification of entries.

Entry Information (M)
A parameter, used in MS abstract operations, which conveys selected information from an entry.

Entry Information Selection (M)
A parameter, used in MS abstract operations, which indicates which attributes from an entry is being requested.

Entry Status (M)
An attribute giving information about the processing status of that entry. Possible values are new, listed or processed.

Entry Type (M)
An attribute which signals if an entry is associated with a delivered message or a delivered report.

Fetch Operation (M)
An MS abstract operation which allows one entry or parts of an entry to be fetched from the stored messages information base.

Fetch Restrictions (M)
Restrictions, imposed by the User Agent, on what kind of messages it is prepared to receive as a result of a fetch operation. The possible restrictions are on message length, content types and EITs.

Filter (M)
A parameter, used in abstract operations, to test a particular entry in an information base

and is either satisfied or not by that entry.

Filter Item (M)
An assertion about the presence or value(s) of an attribute of a particular type in an entry under test. Each such assertion is either true or false.

Forwarding Request (M)
A parameter that may be present in a Message Submission operation, invoked by the User Agent, to request that a message be forwarded from the MS.

Function
A programming language construct modelled after the mathematical concept of function. A function encapsulates some behaviour. It is given some arguments as input, performs some processing and returns some results. Also known as procedure, subprogram or subroutine. See *operation*.

General Attribute (M)
A set of MS attributes which are valid for all types of message and reports independent of content type.

Implementation-Defined
A feature that is not consistent across all implementations; each implementation will provide documentation of behaviour of such a feature.

Indirect Submission (M)
An MS abstract service that offers the same services of the Message Submission service (from the MTS abstract service) with the added functionality of forwarding messages residing in the MS.

Information Base (M)
Objects within the MS which store information relevant to the MS abstract service, e.g., the stored messages information base, which stores the messages and reports that have been delivered into the MS.

Information Base Type (M)
The type of information base; e.g., stored messages, inlog, outlog.

Invoke ID
An integer used to distinguish one (MS) operation from all other outstanding ones.

List Operation (M)
An MS abstract operation which allows a selection of entries from an information base and requested attribute information to be returned for those entries.

Locally Administered
The configuration is not consistent across all systems and the administrator of each system will provide documentation of its behaviour.

Main Entry (M)
For each successful MS abstract operation which creates information base entries, there is always one main-entry; further or more detailed information resulting from the same MS abstract operation can be stored in child-entries.

Matching (M)
The process of comparing the value supplied in an attribute value assertion with the value of the indicated attribute type stored in the MS or deciding whether the indicated attribute type is present.

Message Store (M)
The X.400 standard means by which messages can be delivered by the Message Transfer service into a remote mailbox that is always available and from which a user can retrieve messages at its convenience; in fact, the MS also provides retrieval, indirect submission, and administration services to its user.

MS (M)
Abbreviated form of Message Store.

Multi-valued Attribute (M)
An attribute which can have several values associated with it.

Object (Object) (O)
A composite information entity comprising zero or more OM attributes of different types.

Object Identifier
A value, distinguishable from all other such values, which is associated with an information object.

OM Attribute (O)
An OM attribute comprises one or more values of a particular type (and therefore syntax).

OM Class (O)
A static grouping of OM objects, within a specification, based on both their semantics and their form.

Operation
Processing performed within the MS to provide a service, such as a fetch operation. It is given some arguments as input, performs some processing and returns some results. An application process invokes an operation by calling an interface function.

Original EITs (M)
An attribute identifying the original encoded information types of the message content.

Outstanding Operation
An operation, invoked asynchronously, which has not yet been the subject of a call to **Receive-Result**( ).

Override (M)
A component of the selector parameter indicating that the previously registered restrictions for this abstract operation should not apply for this instance of this abstract operation.

Package (O)
A specified group of related OM classes denoted by an Object Identifier.

Parent Entry (M)
A parent-entry has one or more child-entries which were created as a result of the same abstract operation. If a parent-entry is not a child-entry of another parent-entry, it is a main-entry.

Partial Attribute Request (M)
A component of the *Entry Information Selection* which enables the return of only selected values of a multi-valued attributes.

Private Object (O)
An OM object created in a workspace using the Object Management functions. See *public*

object.

Process
An address space, a single thread of control that executes within that address space and its required system resources. As opposed to a ‘‘system process’’, or the OSI usage of the term ‘‘application process’’. On a system that implements threads, a process is redefined to consist of an address space with one or more threads executing within that address space and their required system resources.

Public Object (O)
A descriptor list which contains all the OM attributes of an OM object. See *private object*.

Register-MS Operation (M)
An MS operation that allows the User Agent to register certain information, that is relevant to interworking between the User Agent and the MS, within the MS.

Registration (M)
Information registered in the MS and stored (until changed by the *Register-MS Operation*) between sessions (or associations).

Result
Information returned from a function or operation and which constitutes the outcome of the processing which was performed.

Returned Content Entry (M)
An *Entry Type* in the stored messages information base which contains the returned content from a previously submitted message.

Selector (M)
A parameter used in functions to select entries from an information base.

Sequence Number (M)
An MS attribute which uniquely identifies an entry within the MS; sequence numbers are allocated in ascending order.

Session
A binding or an abstract association between a particular client and the service over which a series of interface functions are requested.

Single-valued Attribute (M)
An MS attribute that can have only one value associated with it.

Stored Messages (M)
The type of information base used to store entries containing messages and reports delivered by the Message Transfer System to the MS.

Summarize Operation (M)
An MS operation that allows a quick overview of the kind and number of entries which are currently stored in an information base.

Syntax (O)
An OM syntax is any of various categories into which the Object Management Specification statically groups values on the basis of their form. These categories are additional to the OM type of the value.

Thread
A single sequential flow of control within a process.

Undefined
A feature is undefined if this document imposes no portability requirements on applications for erroneous program construct or erroneous data. Implementations may specify the result of using the feature; but such specifications are not guaranteed to be consistent across all implementations. That is, it is always permissible to use the feature but the result is not known unless specified by the particular implementation. See *unspecified*.

Unspecified
A feature is unspecified if this document imposes no portability requirements on applications for correct program construct or erroneous data. Implementations may specify the result of using the feature; but such specifications are not guaranteed to be consistent across all implementations. That is, it is always permissible to use the feature but the result is not known unless specified by the particular implementation. See *undefined*.

Value (O)
See *Attribute Value*.

Workspace (O)
A space in which OM objects of certain OM classes can be created, together with an implementation of the Object Management functions which supports those OM classes.

Index