for Information Systems —

Communication Protocol — Open MUMPS Interconnect

ANSI/MDC X11.2-1995

ANSI/MDC X11.2-1995

American National Standard
for Information Systems —

Communication Protocol — Open MUMPS Interconnect

Sponsor

MUMPS Development Committee

Approved December 11, 1995

American National Standards Institute, Inc.

Abstract

Open MUMPS Interconnect defines a computer communication method for access to MUMPS databases. Implementations of the MUMPS language or other computer languages may use the method, which employs the client-server model for communication over a virtual circuit.

Published by

M Technology Association
1738 Elton Road, Suite 205
Silver Spring, MD 20903
(301) 431-4070

Contents

Foreword

(This foreword is not part of American National Standard ANSI/MDC X11.2-1995.)

Open MUMPS Interconnect defines a computer communication protocol for access to MUMPS databases. Developed primarily for connection of different implementations of the MUMPS language, the protocol may also be used by other languages to gain access to a MUMPS database or to provide a MUMPS database service.

Users of different vendors' MUMPS products may connect them with this protocol, thus extending their networks beyond the limits of proprietary protocols.

Vendors of MUMPS and other database products may provide this protocol to ease their entry into existing networks.

Open MUMPS Interconnect provides all basic operations on the sparse tree-structured MUMPS database through a relatively simple client-server protocol communicating over a virtual circuit.

This standard contains three annexes. Annex A, which is normative and considered part of this standard, prescribes conformance levels and statements. Annexes B and C are informative and not considered part of this standard. Annex B presents the rationale for some of the sponsor's design decisions. Annex C describes possible future extensions to the standard.

The MUMPS Development Committee prepared this document and welcomes suggestions for improvement of the standard. Please submit suggestions to the MUMPS Development Committee, c/o MDC Secretariat, 1738 Elton Road Suite 205, Silver Spring, MD 20903.

The MUMPS Standards Interpretation Review Board, chartered by the MUMPS Development Committee and the M Technology Association but independent of both, reviews and arbitrates disputed interpretations of this standard. Contact the board c/o M Technology Association, 1738 Elton Road Suite 205, Silver Spring, MD 20903.

The following organizations, recognized as having an interest in the standardization of Open MUMPS Interconnect, were contacted prior to the approval of this standard. Inclusion in this list does not necessarily imply that the organization concurred with the submittal of the proposed standard to ANSI.

AICCP
AMS
Atlantic Consultants
Baudet International
Baylor College of Medicine
Branson, Brian
Brigham and Women's Hospital
Carlisle, Terry B.
Children's Hospital Wisconsin
Collaborative Medical Systems
Commercial Data Systems Ltd.
Connections Group
Cunningham, Breck A.
De Moel, Ed
Epic Systems Corp.
Flagship Technologies Inc.
GSC Associates Inc.
HealthPartners
Heffernan, Henry G.
IDX Systems Corp.
Lauteren, Mark
M-Technology
McIntyre Consulting, Inc.
Mformation Systems
Micronetics Design Corp.
MIDS Inc.
National Data Corp.
NIH
Paragon Training
Parr, Walter A.
Polylogics Consultin
Quality Data Service
Rechenzentrum der
Ruh, Lawrence A.
Scicor Inc.
Sentient Systems
Thomas Consulting
UCSF Medical Cente
University Health As
University of Californ
University of Utah Sc
VAISC - DC
VAISC6 - San Franci
VAMC - Seattle
Walker, Kirt
West Virginia Univer
Whitten, David

American National Standard
for Information Systems —

Communication Protocol — Open MUMPS Interconnect

1 Scope, purpose, and application

1.1 Scope

Open MUMPS Interconnect defines a method for network access to MUMPS databases. The protocol provides all basic operations on the sparse tree-structured MUMPS database.

1.2 Purpose

Because the MUMPS language standard (see 2) defines the operations on its database facilities, a MUMPS-specific protocol is required to extend these facilities into an open systems environment.

1.3 Application

Developed primarily for connection of different implementations of the MUMPS language, the protocol may also be used by other languages to gain access to a MUMPS database or to provide a MUMPS database service.

OMI may also be used for inter-task operations on a single computer, for example between two different products.

2 Normative references

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

ANSI X3.4-1990, *American standard code for information interchange*

ANSI/MDC X11.1-1995, *American national standard for information systems — programming languages — MUMPS*

ISO 8859, *Information processing — 8-bit single byte coded graphic character sets — Part 1: Latin alphabet No. 1*

3 Definitions

3.1 MUMPS: name of the programming language defined by ANSI/MDC X11.1.

3.2 OMI: an acronym for Open MUMPS Interconnect.

3.3 OSI: the Open Systems Interconnect model of computer communication protocols.

3.4 message: a string of 8-bit characters, allowing all character codes from 0 to 255.

3.5 circuit: an error-free, sequence-preserving path for messages. The virtual circuit provided by an OSI level 4 service.

3.6 client: a process that originates messages (requests) to be transmitted to a server. Most OMI clients are MUMPS application processes, but a network manager utility program or a non-MUMPS application program could be a client.

3.7 server: a process that responds to clients' requests by performing database functions on their behalf and returning response messages.

3.8 transaction: one request message and the related response message. An OMI transaction is not the same as a transaction processing transaction.

3.9 agent: a process that manages transactions on behalf of one or more clients.

3.10 session: a connection between exactly one agent and exactly one server. When establishing a session, the two processes authenticate each other and negotiate the parameters of transactions to follow.

3.11 user: the human being for whom a client process runs.

3.12 user ID: a number identifying a user for authorization purposes.

3.13 group ID: a number identifying a group of users for authorization purposes.

3.14 OMI node: Usually a node is a computer, but two or more independent implementations might share a computer without sharing their databases, and thus be separate nodes. Conversely, an implementation might occupy several computers connected by a non-OMI method, and thus be one OMI node. Specifically, an OMI node is a set of one or more environments that are accessible without use of the OMI network. That is, a program can refer to an environment in its own node without using OMI.

4 General description

4.1 OMI and MUMPS

ANSI/MDC X11.1 defines the components of the MUMPS database and operations on them. OMI makes some of these operations, listed here, available over a network. The following references are to the formal definitions in ANSI/MDC X11.1; the additional text is for information only:

- *environment:* 7.1.2.4 identifies a specific set of all possible names — a name space. A particular name may appear only once in an environment. Typical implementations place an environment in a directory or a user class, and — with a network — on a node as well.

- *gvn:* 7.1.2.4 Global variables are persistent data records organized into trees. gvn is the name of a global variable, for example:

  ^INV(5321,"Denver","qu")

  A global variable may optionally have a value, and it may optionally have descendants. A tree of global variables is informally called a "global."

- *naked reference:* 7.1.2.4 a shorthand form of gvn referring to the last-used gvn.

- *nref:* 8.2.12 The objects of database lock operations are organized into trees. Their names are similar to gvns, but they relate to gvns only by an application's convention. nrefs have no values.

- *$Data:* 7.1.5.3 Function of a gvn indicates whether the variable it names has a value and whether it has descendants.

- *$Get:* 7.1.5.7 Function of a gvn returns the value of the variable it names, or the empty string if the variable is not defined.

- *$Job:* 7.1.4.10 An unsigned decimal integer uniquely identifies a process on a particular computer.

- *Kill:* 8.2.11 deletes a global variable and its descendants.

- *Lock+:* 8.2.12 claims exclusive use of an nref or a list of nrefs. If the claim succeeds, other concurrent processes' claims on those nrefs will not succeed.

- *Lock-:* 8.2.12 releases a claim on an nref or a list of nrefs.

- *Merge:* 8.2.13 assigns the value of one variable to another and then assigns the values of all the first variable's descendants to corresponding descendants of the second variable.

- *$Order:* 7.1.5.11 Function of a gvn returns the final subscript of the next gvn in a defined tree-walking order.

- *$Query:* 7.1.5.15 Function of a gvn returns the next gvn in a defined tree-walking order.

- *Set:* 8.2.18 creates a global variable and assigns it a value.

- *Set $Extract:* 8.2.18 creates a global variable and modifies its value by assigning a range of its character positions.

- *Set $Piece:* 8.2.18 creates a global variable and modifies its value by assigning one or more of its pieces.

B.6 (of this standard, not X11.1) suggests how implementations may combine these basic operations to achieve more complex MUMPS operations.

4.2 OMI and the OSI network model

OMI provides the services described by levels 5 and 6 of the Open Systems Interconnect model. Level 5, the session layer, involves creating and terminating communication sessions between co-operating systems, while Level 6, the presentation layer, deals with data formats. These OMI services make the network transparent to applications (OSI level 7) for their remote database operations.

In turn, OMI relies on the virtual circuits of an existing OSI level 4 service to provide reliable sequential transmission of messages.

Successful communication with OMI depends on hardware and software to establish compatible protocols at level 4 and below, which are beyond the scope of this standard.

4.3 Client-server protocol

OMI is a connection-oriented protocol based on the client-server model. An OMI node may offer client functions, server functions, or both, as its applications require and its implementer chooses.

4.3.1 Sessions

OMI uses one circuit to establish a session between exactly two OMI nodes. However, implementers may supply servers that support many sessions, to give several client nodes access to the server node's database. Implementers may also permit more than one session on a client node, so that different client processes may refer to data on several server nodes. However, a particular client shall connect to a particular server through only one session at a time, because duplicate paths between a client and a server could cause transactions to be processed out of sequence.

Initiation and termination of sessions shall be invisible to application programs. The implementation shall establish and terminate sessions as necessary to perform OMI operations.

A session is not symmetric. Requests from one client node are satisfied at the other server node. Another session is needed to handle requests in the opposite direction.

4.3.2 The role of the agent

OMI allows, but does not require, an agent process that multiplexes requests from any number of clients on one node in a single session. Thus several clients may share one connection to a server, and all OMI servers shall accept multiplexed requests.

NOTE - Although agents are an important conceptual part of this standard, their existence does not affect the form, content, or sequence of messages. A simple implementation could provide one client per session, and you may read "client" wherever "agent" appears.

This document describes the agent as a separate process, but an implementation may provide its functions by any method.

The agent is synchronous. A session supports only one transaction at a time. The agent shall send one request and then no more until the server's response arrives or the circuit fails.

4.3.3 Transactions

An agent shall originate each transaction with exactly one request message. The server shall attempt the requested operation and shall reply with exactly one response message, which may indicate failure to perform the operation.

NOTE - In anticipation of future versions of the protocol, which may permit many requests and responses in a transaction, messages include sequence numbers and request identifiers.

Only the agent shall initiate a transaction. If the server has something to tell the agent, for example "shutting down," the server shall indicate in its next response message that the agent should request the server's status.

Transactions are isolated. The server need not retain information about completed transactions.

Nonetheless, both server and agent shall retain the parameters of the session including, for example, agent's and server's names, and negotiated maximum lengths of gvns, values, and messages.

All OMI transactions shall have the same priority or the same class. That is, there are no out-of-band messages. Protocol management transactions such as capability negotiation, status updates, or startup and shutdown shall be handled in the same manner as database transactions.

4.3.4 Complex locks

A client may lock a list of nrefs that lie on different OMI nodes. The lock operation is atomic — it shall succeed only if all nrefs in the list are successfully claimed. This property, combined with the isolated transactions between the agent and its servers, requires the agent to assemble a complex lock from OMI lock requests to the respective servers.

The agent shall claim all nrefs of a complex lock. If all the claims succeed, then the entire lock succeeds. Otherwise, the agent shall release its successful claims and repeat the procedure until the lock succeeds or times out.

NOTE - To facilitate this procedure, the protocol defines a request to unlock all nrefs held by a client on one server.

4.4 National character sets

ANSI/MDC X11.1, Annex A, requires ASCII symbols for codes 0 - 127. Many English language terminals and all other language terminals require different encodings, some of which use codes 128 - 255 as well. Most information systems have addressed the question of printing a symbol on a printer that uses a different code from the keyboard that originated the symbol. Networked systems should also address the translation of codes between agents and servers that use different character sets.

All OMI messages shall use the ISO 8859-1 Latin alphabet No. 1, which equals ASCII for codes 0 - 127, and which specifies common European symbols for codes 128 - 255.

Both agents and servers shall send the standard character set on output and shall accept it as input, except for the following special case:

OMI provides an exemption from this requirement for an agent and a server that share a single non-standard character set internally. When establishing their session they may select untranslated messages, thus avoiding the burden of two reciprocal translations.

4.5 Security

Security of computer networks has many facets. This standard addresses some of them and leaves some to other layers of the protocol stack. The general principle is to afford the networked system as much security as typical single-computer commercial systems furnish.

4.5.1 Privacy

OMI offers no privacy protection. System managers who wish to thwart eavesdropping should specify lower-level protocols that provide encryption.

4.5.2 Data Integrity

OMI depends on the virtual circuit of its underlying level 4 service to provide integrity of the data in OMI messages. An OMI transaction either completes or fails entirely, therefore a broken connection cannot cause a partial database operation.

4.5.3 Authentication

When establishing a session, the server and the agent shall exchange passwords to verify each other's authenticity.

4.5.4 Identification

A server shall be identified by one or more names that are available to agents. Likewise, an agent shall be identified by one or more names that are available to servers. The server and agent shall exchange their names when establishing a session.

Each message shall identify the user. Together, the agent's name and the user's identification furnish proof of origin, that is, the server "knows" the origin of a request. Completion of the transaction provides the agent with proof of delivery of the request.

4.5.5 Authorization

OMI supports authorization, the determination that the user has authority for the requested operation. The method encompasses most MUMPS implementations of authorization (see A.3).

The agent shall include a user ID and a group ID in each request. Both identifiers shall pertain to the server, that is, the agent shall translate its implementation's identifiers to the server's identifiers.

NOTE - A different translation may be necessary for each server.

The server shall test the request's user ID and group ID for authorization to perform the requested operation on the object of the request. If either the user ID or the group ID passes the test, then the request is authorized.

The server shall not perform an unauthorized request, but shall return an authorization exception response.

4.6 Replication

Implementers of OMI may offer replication of changes to selected global variables. That is, *set* and *kill* operations are also applied to global variables of the same name in one or more other environments, some of which may reside on other OMI nodes.

Replication is often applied symmetrically to keep the databases identical, whichever one a client changes explicitly. For example, ^XX on computer A is replicated to ^XX on computer B and vice versa, so a change to either one evokes an OMI request to change the other. Then the protocol must avoid an endless series of transactions caused by mutual replication.

Requests (*set*, *set extract*, *set piece*, and *kill*) that alter variables shall include a replication flag to indicate whether the request should be replicated.

NOTE - Implementations of OMI are not required to perform replication. They are required to provide the replication flag in order to cooperate with implementations that do perform replication.

All agents shall originate *set*, *set extract*, *set piece*, and *kill* requests with replication enabled. Servers shall attempt the requested operation whether or not replication is enabled, subject to other considerations like authorization, and shall reply as described in 4.3.3.

An implementation that supports replication shall take further action if and only if it successfully performs a *set*, *set extract*, *set piece*, or *kill* operation with replication enabled. (The operation may have originated from an OMI request or from a local *set*, *set extract*, *set piece*, or *kill* by a process on the server's own node.) The agent on that node shall:

- establish sessions with the replication destination nodes, if not already established.

- send the same request with replication disabled to each of the destinations, in effect saying, "This is a replication message. Do not replicate it further."

4.7 Environments

Agents shall translate the client's environments for gvns and nrefs to the server's environments, defined in 4.1. The translation may be implicit — hidden from the client program with a translation table defined by the network manager — or explicit through extended references that include an environment specification in the client program.

In either case, the implementation shall not require the client program to contain any information about the server's name or environments. If necessary, the agent shall translate the client's environment specification to a server's name and to an environment as it is known on the server's node.

The server may treat a request whose environment equals an empty string as an error, or it may apply a default environment.

4.8 OMI version negotiation

In anticipation of future versions of OMI, some of which may not be compatible with earlier versions, the agent and server shall negotiate a commonly supported OMI version when establishing a session. This negotiation permits most new releases of OMI implementations to be installed one node at a time.

If the agent supports multiple OMI versions, it shall send connect requests for each until the server accepts one. 5.3.1 shows how the location and format of the version number in the connect requests of all versions are made accessible to servers of all versions.

4.9 Exception handling

OMI defines four categories of exceptions: those that a program might expect from its database operations, those related to the protocol itself, errors during session establishment, and errors from the virtual circuit. See 5.3.2 for details of error codes.

Some errors are fatal to OMI and require that a new session be established. When processing these fatal errors, the implementer may choose to maintain or to disconnect the underlying virtual circuit. In either case, the establishment of a new OMI session ensures that subsequent transactions are valid.

4.9.1 Database errors

Implementers may choose to handle errors arising from database operations in the same way whether the database is local or remote. OMI responses indicate errors like undefined global variable, unauthorized operation, reference to an undefined environment, or maximum length exceeded. OMI does not support the MUMPS naked reference because that would require the server to retain information about past transactions. Agents shall detect naked reference errors.

OMI does not specify the timing of responses to database update requests. Servers that defer disk writing may be implemented to notify their own system manager of a failure to write, and not to delay their OMI response until the disk operation completes.

4.9.2 OMI protocol exceptions

OMI responses indicate protocol failures like illegal requests, sequence errors, or negotiated limits exceeded. OMI also provides a "service suspended" response for occasions like database backup where the request might succeed if repeated later.

4.9.3 Session establishment errors

A *connect* transaction must succeed before other requests are valid. If the *connect* fails or is not sent by the agent at the appropriate time, then no session is established.

4.9.4 Virtual circuit failure

The underlying level 4 services used by the agent and server may report failure of the virtual circuit. Such failures are fatal to OMI; the session must be reestablished.

NOTE - Implementers may choose to treat circuit failures as hardware errors, notifying the affected clients and logging the events for corrective action. However, this is not required; termination of a session need not affect client operations. The agent may choose to defer pending requests, reestablish the session, and continue processing, without delivering any errors to the application level.

4.10 Implementation limits and portability

When establishing a session, the agent and server shall negotiate the maximum mutually acceptable sizes of a gvn, its value, an nref, and the messages that contain them. The agent offers its minimum and maximum for each. The server compares these with its own minima and maxima, then replies with the maxima to be used during the session.

Agents and servers must support at least the portability requirements of ANSI/MDC X11.1 and the minimum message sizes they imply. The OMI protocol permits larger limits by negotiation, so implementations that exceed the portability requirements can use longer fields. OMI does have its own rather large limits; clause 5 describes them.

4.11 Extensions to the standard

OMI admits extensions for specific implementations or for specific applications. Extensions to the standard may afford increased performance in homogeneous networks, add proprietary functions, or enable experimentation with features planned for new versions of OMI.

Implementers shall register their extensions with the MUMPS Development Committee, the sponsor of ANSI/MDC X11.2, to assure uniqueness. Two numbers identify an extension; an operation class and an error class. Each class comprises 65 536 values.

An extension may coexist with other extensions. The agent and the server shall notify each other of their available extensions when establishing the session, and they may use any number of these extensions during the session.

Implementers define extensions, and they shall bear responsibility for conflicts with the standard and among extensions. An implementation that provides an extension shall also provide the standard protocol.

5 Message form and content

5.1 Characters

OMI messages consist of 8-bit bytes (octets). No characters have special meaning to the protocol, thus the bytes may assume any value from 0 to 255. Although negotiations between the agent and server may limit the range of valid characters in particular fields, the underlying circuit shall transport all 8-bit bytes. See 4.4 for more about character sets.

5.2 Fields

OMI messages are defined in terms of these 6 fields:

- *Short integer <SI>:* 1 byte represents an unsigned integer ranging from 0 to 255.

- *Long integer <LI>:* 2 bytes represent an unsigned integer ranging from 0 to 65 535, with the low order byte sent first (little endian).

- *Very long integer <VI>:* 4 bytes represent an unsigned integer ranging from 0 to 2^{32}-1. The bytes are sent in order of increasing significance, from low to high.

- *Short string <SS>:* a counted string comprising a 1 byte length <SI> followed by 0 to 255 bytes of data.

- *Long string <LS>:* a counted string comprising a 2 byte length <LI> followed by 0 to 65 535 bytes of data.

- *Very long string <VS>:* a counted string comprising a 4 byte length <VI> followed by 0 to 2^{32}-1 bytes of data.

5.3 Message form

OMI messages comprise a header followed by 0 or more of the fields described in 5.2. The 2 types of OMI messages — request and response — have different headers. The number and form of the following fields depend on the operation class and type. The overall message is a very long string <VS> so that message boundaries can be clearly distinguished.

The total number of bytes in a message shall not exceed 65 535.

5.3.1 Request header

Each request shall contain a header that, among other things, identifies the user. Inclusion of this information permits transactions to be isolated (see 4.3.3). Without it the server would have to track users and their authorizations across transactions.

The request header is a short string <SS> so that the server can locate the field following the header regardless of the length of the header.

NOTE - The request header has fixed length, but future versions of OMI may specify different lengths. Servers and agents need to find the version numbers following the header in connect messages, regardless of the version in use (see 4.8).

The fields of the request header and their sequence shall be:

a *Operation class:* <LI> denotes 1 of 65 536 classes of operations. The MUMPS Development Committee assigns classes to OMI versions and to specific extensions of the standard (see 4.11). The class of operations specified by ANSI/MDC X11.2 shall be 1. All other classes are reserved for assignment by the MUMPS Development Committee.

b *Operation type:* <SI> denotes a specific operation within an operation class. For example, *set* and *kill* are operation types 10 and 13.

c *User identifier:* <LI> identifies a user for security purposes. See 4.5.

d *Group identifier:* <LI> identifies a group of users for security purposes. See 4.5.

e *Sequence number:* <LI> This sequential number starts at 1 and increments by 1 to 65 535, then continues with 1. The server may use this number to verify that requests are received and processed in the correct order. The agent may choose any valid starting sequence number for a *connect* request, causing the server to synchronize at that sequence number.

f *Request identifier:* <LI> links responses to requests. The content of this field is left to the implementer, except that the server must return the value from each request in the corresponding response.

Table 1 shows the OMI operations and the values assigned to their operation types. These operations suffice for the database functions of ANSI/MDC X11.1. The *unlock client* and *unlock all* operations provide additional convenience for implementers.

```
+-------------------------------------------+
¦                  <VS>                     ¦
¦ ááááá +-----------------------------------¦
¦ <VI>  ¦  Header  ¦   Operation Specific Fields   ¦
+-------------------------------------------+
```

Figure 1 - Form of a message

```
+-------------------------------------------------------------+
¦                          <SS>                               ¦
¦        +----------------------------------------------------¦
¦ <SI>   ¦ <LI>  ¦  <SI> ¦  <LI> ¦  <LI>  ¦   <LI>   ¦  <LI>  ¦
¦        ¦ Class ¦  Type ¦  User ¦  Group ¦ Sequence ¦ Req ID ¦
+-------------------------------------------------------------+
```

Figure 3 - Form of a request header

Control operations establish and maintain the OMI session. They do not relate to application-level processes.

The global update and fetch operations perform assignment and retrieval of global variables and their values. They correspond directly to application-level database operations.

Lock operations claim and release exclusive use of <u>nref</u>s, which correspond to database records by application program conventions.

5.3.2 Response header

Each response shall contain a header that describes the server's response to one request.

The response header is a short string <SS> so that the agent can locate the field following the header. 5.3.1 discusses the need for this ability.

Table 1 - Operation types

Type	Name	Description
		Control operations
1	Connect	Creates a session between an agent and a server. A connect request shall be sent by an agent and accepted by the server before the server accepts any other messages.
2	Status	Requests status from the server. Sent by the agent whenever server status is required.
3	Disconnect	Terminates an OMI session. A disconnect request should be sent by the agent on shutdown, before disconnecting the underlying circuit.
		Global update operations
10	Set	Assigns a value to a global variable.
11	Set Piece	Assigns a value to a piece of a global variable.
12	Set Extract	Assigns a value to positions within a global variable.
13	Kill	Kills a global variable.
		Global fetch operations
20	Get	Requests the value of a global variable. Returns the value, or the empty string if the variable is undefined. A status flag indicates if the variable was defined, so that the agent can use this message for either the MUMPS $Get function or a normal global fetch.
21	Define	Requests the status of a global variable.
22	Order	Requests the following global subscript or the following global name in collating sequence. Returns the empty string if no following subscript or global name is defined.
24	Query	Requests the next gvn in collating order. Returns the empty string if no further gvns are defined.
25	Reverse order	Requests the preceding global subscript or the preceding global name in collating order. Returns the empty string if no preceding subscript or global name is defined.

Table 1 - Operation types *(Concluded)*

Type	Name	Description
		Lock Operations
30	Lock	Claims an incremental lock on an nref.
31	Unlock	Releases a lock on an nref.
32	Unlock client	Unlocks all nrefs for one client.
33	Unlock all	Unlocks all nrefs for all clients on the client node.

The fields of the response header and their sequence shall be:

a *Error class:* <LI> denotes 1 of 65 536 classes of errors. The MUMPS Development Committee assigns classes to OMI versions and to specific extensions of the standard (see 4.11). The classes of errors specified by ANSI/MDC X11.2 shall be 0 and 1. All other classes are reserved for assignment by the MUMPS Development Committee.
An error class of 0 means that the request completed successfully. Class 1 indicates failure to perform the request.

b *Error type:* <SI> indicates the result of the server's attempt to perform the request. When the error class is 0, indicating success, then the error type indicates types of success. For other error classes, the error type denotes a specific error condition.

c *Error modifier:* <LI> modifies the meaning of specific error conditions. This field is optional, therefore it may be set to 0 by a server and may be ignored by an agent. See specific operations in 5.4 for use of the error modifier.

d *Server status:* <LI> conveys information about the server itself, and is not necessarily related to the request. Each change in server status is conveyed in only one response. A value of 0 means that no change in server status has occurred since the last response.

e *Sequence number:* <LI> This number shall equal the sequence number of the corresponding request.

f *Request identifier:* This number shall equal the request identifier of the corresponding request.

Table 2 shows the error conditions that the server shall indicate in responses when applicable. The error class field shall equal 1 in all error responses where a numeric value for the error type appears.

Table 2 also shows other errors for completeness. Circuit errors by definition cause failure to deliver OMI messages. Database exception conditions are not protocol errors. Their error class equals 0 and they are signaled in other fields of the response.

NOTE - The grouping of error types in table 2 and the assignment of their numeric values is for didactic convenience and does not imply any special qualities or processing requirements.

```
+-------------------------------------------------------------------+
¦                              <SS>                                 ¦
¦     +-------------------------------------------------------------¦
¦<SI> ¦ <LI>  ¦ <SI> ¦  <LI>    ¦     <LI>      ¦   <LI>   ¦  <LI>  ¦
¦     ¦ Class ¦ Type ¦ Modifier ¦ Server Status ¦ Sequence ¦ Req ID ¦
+-------------------------------------------------------------------+
```

Figure 5 - Form of a response header

Table 3 - Error conditions

Type	Name	Description
		Database errors
—	Undefined global	Signaled in the Define field of the Get response message.
—	Lock not granted	Signaled in the Lock Granted field of the Lock response message.
1	User not authorized	The user did not have authorization to perform the requested operation.
2	No such <u>environment</u>	The <u>environment</u> in the global reference is not known to the server.
3	Global reference content not valid	The global reference contains invalid characters.
4	Global reference too long	The global reference is longer than the maximum negotiated for this session.
5	Value too long	The value specified is longer than the maximum negotiated for this session.
6	Unrecoverable error	The server encountered a fatal error while processing the request, for example, disk full or write error.
		Protocol errors
10	Global reference format not valid	The structure of the global reference is incorrect. This error indicates a failure in the agent or server logic.
11*)	Message format not valid	The format of the message is incorrect.
12	Operation type not valid	The operation type requested by the agent is not known to the server.
13	Service temporarily suspended	The server has temporarily suspended OMI operations. The agent may choose to disconnect or to retry the operation later. This error is intended for use during backup or other maintenance operation.
14*)	Sequence number error	The server received a request with a sequence number that did not follow the sequence number of the previous request. Indicates either a logic failure or lost message(s).

Table 2 - Error conditions *(Concluded)*

Type	Name	Description
		Session Establishment errors
20	OMI version not supported	The OMI major version requested by the agent in the connect message is not supported on the OMI server. The agent may try again with another ver sion number.
21*)	Agent min length > server max length	The minimum length required by the agent for value, subscripts, reference or message was greater than the maximum supported by the server. No session can be established.
22*)	Agent max length < server min length	The maximum length allowed by the agent for a field is less than the minimum supported by the server. No session can be established.
23*)	Connect request received during session	The agent sent a connect request to the server when an OMI session was already established.
24	OMI session not established	The agent sent a request to the server before send-ing the connect request.
		Circuit Errors
—	Broken session	
—	Interface busy	
—	Insufficient resources	
*) These errors are fatal to the session. The agent shall reestablish the session.		

5.3.3 Global reference

4.1 describes the environment, the global variable name (gvn) and the lock argument (nref) that appear in many OMI messages, where they have the following form, called a global reference.

A global reference shall be a long string <LS>. Its fields shall be strings containing the environment, global name and subscripts.

NOTE - The protocol does not support MUMPS naked references. The agent shall send full refer-ences for all operations.

The fields of the global reference and their se-quence shall be:

a *Environment:* <LS> denotes the server's environment for this global reference. See 4.1 and 4.7.

b *Name:* <SS> The name shall include a lead-ing caret as shown in 4.1.

c *Subscript(s):* <SS> 0 or more subscripts.

```
+----------------------------------------------------------------+
¦      <LS>           ¦    <SS>    ¦    <SS>     ¦    <SS>     ¦     ¦
¦ <LI> ¦ Environment  ¦<SI>¦ Name  ¦<SI>¦ Sub 1  ¦<SI>¦ Sub 2  ¦ ... ¦
+----------------------------------------------------------------+
```

Figure 7 - Form of global reference

Each shall be a short string, thus limited to 255 characters. Numeric subscripts shall appear as ASCII characters, not as binary numbers or other internal representation.

NOTE - The number of subscripts is implied by the length of the entire global reference, a long string <LS>.

5.4 Requests and responses

A complete request or response message is a <VS>, whose string comprises the request or response header followed by operation-specific fields defined here.

5.4.1 Connect

(Operation type 1) The fields of the *connect* request and their sequence shall be:

a *Major version:* <SI> the major version of the OMI protocol supported by the agent. This document defines major version 1. If the agent supports multiple versions, it shall send *connect* requests for each until the server accepts a major version.

b *Minor version:* <SI> the minor version of the OMI protocol supported by the agent. This document defines minor version 1. The agent should send the highest minor version it supports.

c *Min/max values:* <LI> The next 10 fields are 5 pairs of minimum and maximum lengths for global values, individual subscripts, global references, total message length, and limit on requests outstanding. These values represent the minimum acceptable to the agent and the maximum that can be processed by the agent.

The server shall compare these values with its own limits and shall return either the largest value possible or, if the agent's lengths are not within its bounds, an error condition.

Agents shall send minimum values that accommodate the portability requirements of ANSI/MDC X11.1. Maximum values may be as high as the agent can support. For example, an agent that supports global value lengths of 510 bytes would send min = 255 and max = 510. A server with maximum global value length of 255 would respond with a maximum of 255. The agent should thereafter return a maximum length error to any client that sets global values longer than 255.

Minimum and maximum values for the limit on requests outstanding shall both equal 1. Larger values allow for higher performance in extensions and future versions of the protocol.

d *8 bit:* <SI> The 8 bit flag indicates whether the agent supports 8 bit bytes for global values and subscripts. 0 means that only 7 bit values are valid; 1 means that all 8 bit values are valid.

NOTE - This flag does not affect the overall protocol, which shall transmit 8 bit characters.

e *Character translation:* <SI> The character translation flag indicates whether the agent wishes to send characters in an untranslated form. 1 means do not translate; 0 means translate to the standard character set. See 4.4.

f *Implementation identification:* <SS> identifies the implementation of MUMPS or other software that is running on the agent. This field is for information only and does not affect the operation of the protocol. Implementers shall register their implementation identifiers with the MUMPS Development Committee, the sponsor of ANSI/MDC X11.2, to avoid duplication. Implementers who do not wish to register an identification may use an empty string.

g *Agent name:* <SS> the OMI node name assigned to the agent's node, used along with the

```
 +------------------------------------------------------------+
 ¦<SI> ¦<SI> ¦<LI>¦<LI>¦<LI>¦<LI>¦<LI>¦<LI>¦<LI>¦<LI>¦<LI>¦<LI>¦<SI> ¦
 ¦Major¦Minor¦Min ¦Max ¦Min ¦Max ¦Min ¦Max ¦Min ¦Max ¦Min ¦Max ¦     ¦
 ¦Ver  ¦Ver  ¦  Value   ¦Subscript¦Reference¦ Message ¦Outstand ¦8 bit¦
 +------------------------------------------------------------+
+--------------------------------------------------------------+
¦<SI> ¦    <SS>     ¦    <SS>    ¦    <SS>      ¦    <SS>    ¦<SI> ¦<LI>¦<LI>¦
¦Char ¦<SI>¦Implem¦<SI>¦Agent¦<SI>¦Agent   ¦<SI>¦Server¦Ext  ¦1st ¦2nd ¦
¦Trans¦    ¦ ID   ¦    ¦Name ¦    ¦Passwrd¦    ¦ Name ¦Count¦Ext#¦Ext#¦
+--------------------------------------------------------------+
```

Figure 9 - Connect request

agent's password to authenticate that the agent is not an impersonator.

h *Agent password:* <SS> a password assigned by the network manager, used by the server for authentication.

i *Server name:* <SS> the OMI node name assigned to the server's node with which the agent wishes to establish a session. This name may or may not relate to the names used to establish the circuit in the underlying protocol.

j *Extension count:* <SI> the count of extension numbers that follow.

k *Extension number(s):* 0 or more fields contain extension numbers that the agent proposes to use in this session. These extension numbers are assigned by the MUMPS Development Committee and need not relate directly to operation classes. See 4.11.

The fields of the *connect* response and their sequence shall be:

a *Major version:* <SI> the major version of the OMI protocol supported by the server.

b *Minor version:* <SI> the minor version of the OMI protocol supported by the server. The server shall send the highest minor version it supports that is not greater than the agent's minor version.

c *Max values:* <LI> The next 5 fields are the maximum lengths for global values, individual subscripts, global references, total message length, and limit on requests outstanding. These values shall equal the largest acceptable to the server but shall not exceed the maxima proposed by the agent in the connect request. If the server cannot satisfy this constraint, then it shall return an error condition.

NOTE - When the agent requests minima that meet the portability requirements of ANSI/MDC X11.1, the server should be able to accept those values. Failure to agree means that one of the parties does not conform to the OMI protocol.

The limit on requests outstanding shall equal 1. Larger values allow for higher performance in extensions and future versions of the protocol.

d *8 bit:* <SI> The 8 bit flag indicates whether the server agrees to use 8 bit bytes for global values and subscripts. 0 means that only 7 bit values are valid; 1 means that all 8 bit values are valid.

e *Character translation:* <SI> The character translation flag indicates whether the server agrees to send characters in an untranslated form. 1 means do not translate; 0 means translate to the standard character set. See 4.4.

f *Implementation identifier:* <SS> identifies the implementation of MUMPS or other software that is running on the server. See the corresponding field in the *connect* request.

g *Server name:* <SS> the OMI node name assigned to this server's node.

h *Server password:* <SS> a password assigned by the network manager, used by the agent for authentication.

i *Extension count:* <SI> the count of extension numbers that follow.

j *Extension number(s):* <LI> 0 or more fields contain extension numbers that the server agrees to use in this session. Only extensions proposed by the agent shall be returned. These extension numbers are assigned by the MUMPS Development Committee and need not relate directly to operation classes. See

```
+---------------------------------------------------------------------+
¦ <SI>  ¦ <SI>  ¦ <LI> ¦ <LI> ¦ <LI> ¦ <LI> ¦ <LI> ¦ <SI>  ¦ <SI>  ¦
¦ Major ¦ Minor ¦ Max  ¦ Max  ¦ Max  ¦ Max  ¦ Max  ¦ 8 bit ¦ Char  ¦
¦Version¦Version¦ Value¦ Sub  ¦ Ref  ¦ Mes  ¦ Out  ¦       ¦ Trans ¦
+---------------------------------------------------------------------+
+---------------------------------------------------------------------+
¦     <SS>      ¦     <SS>     ¦     <SS>         ¦<SI>  ¦<LI> ¦<LI> ¦
¦ <SI> ¦ Implem ¦ <SI>¦Server ¦ <SI> ¦ Server    ¦Ext   ¦1st  ¦2nd  ¦
¦      ¦ ID     ¦     ¦Name   ¦      ¦ Password  ¦Count ¦Ext# ¦Ext# ¦
+---------------------------------------------------------------------+
```

Figure 11 - Connect response

4.11.

5.4.2 Status

(Operation type 2) The *status* request consists of a header only.

The *status* response consists of a header only, with status information conveyed in the error type, error modifier, and server status fields.

5.4.3 Disconnect

(Operation type 3) The *disconnect* reason field is an arbitrary text string <LS> that the server should log and display for the network manager.

The *disconnect* response consists of a header only, indicating if an error occurred during the disconnect.

5.4.4 Set

(Operation type 10) This operation assigns a value to a global variable, creating the variable if it does not already exist.

The fields of the *set* request and their sequence shall be:

a *Replicate flag:* <SI> indicates whether the server should replicate the operation (see 4.6). 0 means no, 1 means yes. All other values are reserved.

NOTE - Servers are not required to support replication. Those that do support it shall honor this field.

b *Global reference:* <LS> the name of the global variable (gvn) that is to be set to a new value.

c *Global value:* <LS> The value to be assigned to the global variable.

The *set* response consists of a header only, indicating whether an error occurred during the operation.

5.4.5 Set piece

(Operation type 11) This operation assigns a value to 1 or more contiguous pieces of a global variable's value. ANSI/MDC X11.1, 8.2.18 describes the semantics.

The first 3 fields of this request are identical to those of the *set* request in 5.4.4. These additional fields follow the first 3:

d *Start piece:* <LI> the number of the first piece whose value is to be assigned.

e *End piece:* <LI> the number of the last piece whose value is to be assigned. When assigning a single piece of the value, this field shall equal the start piece field.

f *Delimiter:* <SS> The delimiter of pieces in the value.

The *set piece* response consists of a header only, indicating whether an error occurred during the operation.

```
+--------------------+
¦       <LS>         ¦
¦  <LI> ¦  Reason    ¦
+--------------------+
```

Figure 13 - Disconnect request

```
+--------------------------------------------------------+
¦  <SI>      ¦      <LS>          ¦      <LS>            ¦
¦ Replicate ¦ <LI> ¦ Global Ref ¦ <LI> ¦ Global Value  ¦
+--------------------------------------------------------+
```

Figure 15 - Set request

```
+-----------------------------------------------------------------------+
¦   <SI>    ¦    <LS>         ¦    <LS>          ¦<LI> ¦<LI>¦   <SS>       ¦
¦Replicate¦<LI>¦Global Ref¦<LI>¦Global Value¦Start¦End ¦<SI>¦Delimiter¦
+-----------------------------------------------------------------------+
```

Figure 17 - Set piece request

5.4.6 Set extract

(Operation type 12) This operation assigns a value to 1 or more contiguous character positions of a global variable's value. If the variable does not already exist, the operation creates the variable with a value equal to the empty string before assigning the value. ANSI/MDC X11.1, 8.2.18 describes the semantics.

The first 3 fields of this request are identical to those of the *set* request in 5.4.4. These additional fields follow the first 3:

d *Start position:* <LI> the first position where the value is to be assigned.

e *End position:* <LI> the last position where the value is to be assigned. When assigning a single character position, this field shall equal the start position field.

The *set extract* response consists of a header only, indicating whether an error occurred during the operation.

5.4.7 Kill

(Operation type 13) This operation deletes a global variable and all its descendants.

The fields of the *kill* request and their sequence shall be:

a *Replicate flag:* <SI> indicates whether the server should replicate the operation (see 4.6).

NOTE - Servers are not required to support replication. Those that do support it shall honor this field.

b *Global reference:* <LS> the global reference that is to be killed.

The *kill* response consists of a header only, indicating whether an error occurred during the operation.

5.4.8 Get

(Operation type 20) This operation fetches the value of a global reference given in its global reference field <LS>.

```
+------------------------------------------------------------------+
¦  <SI>    ¦      <LS>           ¦       <LS>            ¦ <LI>   ¦ <LI> ¦
¦Replicate¦ <LI> ¦ Global Ref ¦ <LI> ¦ Global Value¦ Start ¦ End  ¦
+------------------------------------------------------------------+
```

Figure 19 - Set extract request

```
+--------------------------------+
¦  <SI>     ¦       <LS>          ¦
¦ Replicate ¦  <LI> ¦ Global Ref ¦
+--------------------------------+
```

Figure 21 - Kill request

0 means no, 1 means yes. All other values are reserved.

The fields of the *get* response and their sequence shall be:

a *Define:* <SI> indicates whether the global variable has a value. 0 means no, and 1 means yes. All other values are reserved. This distinguishes an empty string value from undefined.

NOTE - A global variable may have descendants without having a value.

b *Global data:* <LS> the value of the requested global variable. If the variable has no value, this field shall equal the empty string.

5.4.9 Define

(Operation type 21) The *define* operation indicates whether the global variable given in its global reference field <LS> has a value and whether it has descendants.

The one field of the *define* response shall be a short integer <SI>, the $DATA value of the global reference. ANSI/MDC X11.1, 7.1.5.3, defines the $DATA values that indicate the presence of a global value and descendants.

5.4.10 Order

(Operation type 22) The *order* operation fetches the following subscript or the following global name, depending on the global reference in its request. ANSI/MDC X11.1, 7.1.5.11 defines the ordering sequence.

5.4.10.1 Subscript

If the global reference <LS> of its request contains one or more subscripts, the *order* operation fetches the following subscript on the same level as the final subscript in the global reference.

NOTE - The final subscript of the global reference may be the empty string, requesting the first subscript at that level in the tree.

The *order* response shall comprise 1 field, a short string <SS> containing the following subscript. If there is no following subscript, this field shall equal the empty string.

5.4.10.2 Global name

If the global reference <LS> of its request contains no subscripts, the *order* operation returns the following global name in the environment.

NOTE - The entire global reference may equal the empty string, requesting the first global name in the environment.

The *order* response shall comprise 1 field, a short string <SS> containing the following global name. If there is no following global name, this field shall equal the empty string.

5.4.11 Reverse order

(Operation type 25) The *reverse order* operation fetches the subscript or the global name preceding the subscript or the global name in the

```
+-----------------------------+
¦       <LS>                  ¦
¦  <LI> ¦ Global Reference    ¦
+-----------------------------+
```

Figure 27 - Define request

```
+----------+
¦  <SI>    ¦
¦  Define  ¦
+----------+
```

Figure 29 - Define response

```
+-----------------------------+
¦       <LS>                  ¦
¦  <LI> ¦ Global Reference    ¦
+-----------------------------+
```

Figure 31 - Order request

```
+--------------------------+
¦     <SS>                 ¦
¦ <SI> ¦ Subscript or Name ¦
+--------------------------+
```

Figure 33 - Order response

```
+-----------------------------+
¦       <LS>                  ¦
¦  <LI> ¦ Global Reference    ¦
+-----------------------------+
```

Figure 23 - Get request

```
+--------------------------------+
¦  <SI>  ¦       <LS>            ¦
¦ Define ¦  <LI> ¦ Global Value  ¦
+--------------------------------+
```

Figure 25 - Get response

global reference <LS> of its request, depending on the form of that global reference. ANSI/MDC X11.1, 7.1.5.11 defines the ordering sequence.

5.4.11.1 Subscript

If the global reference <LS> of its request contains one or more subscripts, the *reverse order* operation fetches the preceding subscript on the same level as the final subscript in the global reference.

NOTE - The final subscript of the global reference may be the empty string, requesting the last sub-script at that level in the tree.

The *reverse order* response shall comprise 1 field, a short string <SS> containing the preceding subscript. If there is no preceding subscript, this field shall equal the empty string.

5.4.11.2 Global name

If the global reference <LS> of its request contains no subscripts, the *reverse order* operation returns the preceding global name in the environment.

NOTE - The entire global reference may equal the empty string, requesting the last global name in the environment.

The *reverse order* response shall comprise 1 field, a short string <SS> containing the preceding global name. If there is no preceding global name, this field shall equal the empty string.

5.4.12 Query

(Operation type 24) The *query* operation fetches the entire global reference following the global reference <LS> of its request. The final subscript of the requested global reference may be the empty string.

The *query* response shall comprise 1 field, a long string <LS> containing the following global refer-ence. If there is no following global reference, this field shall equal the empty string.

5.4.13 Lock

(Operation type 30) The *lock* operation claims ex-clusive use of an nref, which has the form of a global reference. 4.3.4 describes the coordination of multiple lock operations.

The fields of the *lock* request and their sequence shall be:

a *Global reference:* <LS> the reference being claimed.

b *Client identifier:* <SS> $Job (see ANSI/MDC X11.1, 7.1.4.10) of the client pro-cess making the request, an ASCII string of decimal digits.

```
+------------------------------+
¦       <LS>                   ¦
¦  <LI> ¦ Global Reference     ¦
+------------------------------+
```

Figure 39 - Query request

```
+---------------------------+
¦      <LS>                 ¦
¦ <LI> ¦ Global Reference   ¦
+---------------------------+
```

Figure 41 - Query response

```
+----------------------------------------------------+
¦       <LS>                ¦       <SS>             ¦
¦  <LI> ¦ Global Reference  ¦  <SI> ¦ Client ID      ¦
+----------------------------------------------------+
```

Figure 43 - Lock request

```
+--------------+
¦     <SI>     ¦
¦ Lock Grant   ¦
+--------------+
```

Figure 45 - Lock response

```
+------------------------------+
¦       <LS>                   ¦
¦  <LI> ¦ Global Reference     ¦
+------------------------------+
```

Figure 35 - Reverse order request

```
+----------------------------+
¦      <SS>                  ¦
¦ <SI> ¦ Subscript or Name   ¦
+----------------------------+
```

Figure 37 - Reverse order response

The *lock* response contains 1 field indicating whether the claim succeeded. 0 means no, and 1 means yes. All other values are reserved.

5.4.14 Unlock

(Operation type 31) The *unlock* operation releases 1 claim on an nref, which has the form of a global reference.

The fields of the *unlock* request and their sequence shall be:

a *Global reference:* <LS> the reference being unlocked.

b *Client identifier:* <SS> $Job (see ANSI/MDC X11.1, 7.1.4.10) of the client process making the request, an ASCII string of decimal digits.

The *unlock* response consists of a header only, indicating whether an error occurred during the operation.

5.4.15 Unlock client

(Operation type 32) The *unlock client* operation releases all claims on all nrefs held by a client process. This operation implies that the server node's lock manager shall retain the client identifier and node for each lock (see B.4). The client ID field <SS> equals $Job (see ANSI/MDC X11.1, 7.1.4.10) of the client process making the request, an ASCII string of decimal digits.

```
+---------------------------------------------+
¦      <LS>                ¦      <SS>        ¦
¦  <LI> ¦ Global Reference  ¦  <SI> ¦ Client ID ¦
+---------------------------------------------+
```

Figure 47 - Unlock request

The *unlock client* response consists of a header only, indicating whether an error occurred during the operation.

5.4.16 Unlock all

(Operation type 33) The *unlock all* request consists of a header only, indicating that all locks held by all clients on the client node shall be unlocked.

NOTE - This operation is included for the convenience of implementers — there is no requirement that the request be sent in any specific condition.

The *unlock all* response consists of a header only, indicating whether an error occurred during the operation.

```
+--------------------+
¦       <SS>         ¦
¦  <SI> ¦ Client ID  ¦
+--------------------+
```

Figure 49 - Unlock client request

Annex A

(Normative)

Conformance

This annex defines levels of conformance to this standard and prescribes information that implementers shall provide to document the conformance of an implementation.

NOTE - The term *program* used in this annex does not refer to a MUMPS program or routine. It refers to the program with which a MUMPS or other product implements the protocol defined in this standard.

A.1 Implementations

A *conforming implementation* shall

- correctly execute all messages conforming to both this standard and the implementation-defined extensions to this standard;

- reject all messages that contain errors, where such error detection is required by this standard;

- be accompanied by a document which provides a definition of all implementation-defined extensions and a conformance statement of the form:

xxx version *v* implements X11.2-*yyyy* with the following conformance specification:

followed by a table or equivalent presentation of the features shown in table A.1 with the implementation's options or values supplied.

An *MDC conforming implementation* shall be a conforming implementation except that the conforming document shall be this standard together with any such current MDC documents that the vendor chooses to implement. The conformance statement shall be of the form:

xxx version *v* implements X11.2-*yyyy* as modified by the following MDC documents:

ddd (MDC status *sss*)

with the following conformance specification:

followed by a table or equivalent presentation of the features shown in table A.1 with the implementation's options or values supplied.

An *MDC strictly conforming implementation* is an MDC conforming implementation whose MDC modification documents only have MDC Type A status.

A *<National Body> conforming implementation* is an implementation conforming to one of the above options in which any requirements are replaced by the <National Body> requirements and other extensions required by the <National Body> are implemented.

An implementation may claim more than one level of conformance if it provides a switch by which the user is able to select the conformance level.

A.2 Programs

A *strictly conforming program* shall use only the constructs specified in this standard, shall not exceed the limits and restrictions specified and shall not depend on extensions of an implementation.

A *strictly conforming <National Body> program* is a strictly conforming program, except that any limits and restrictions are replaced by those specified by the <National Body> and other extensions required by the <National Body> may be used.

A *conforming program* is one that is acceptable to a conforming implementation.

Table A.1 - Conformance

Clause	Feature	Option/Value
	Hardware Name	
	Hardware Version	
	Operating System Name	
	Operating System Version	
	Special configuration requirements	
4.2	OSI level 4 services required	
4.3	Client?	
4.3	Server?	
4.3.1	Maximum number of client sessions	
4.3.1	Maximum number of server sessions	
4.3.2	Agent provided?	
4.4	Alternative character sets provided	
4.5.5	Translation mechanism for user ID and group ID	
4.6	Replication supported?	
4.7	Environment mapping facility	
4.8	OMI versions supported	
4.9.1	Deferred write error handling	
4.9.4	Virtual circuit error handling	
4.10	Minimum and maximum sizes of the following:	
	gvn	
	gvn value	
	nref	
	messages containing the above	
4.11	Extension numbers	
5.4.1.f	Implementation identifier	

Annex B

(Informative)

Rationale

B.1 Implementation

Although this standard defines a protocol that supports operations on a MUMPS database, the standard does not say how the protocol should be implemented, nor how it should be managed. However, the designers of OMI certainly considered these factors. This annex describes some of their considerations to help you understand, build, or operate an OMI network.

Those planning to implement this standard are advised to study the ANSI/MDC X11.1 standard for complete semantics of the database operations.

B.2 Network management

This standard requires some items whose provision is outside the scope of the standard, for example names and passwords appearing in its messages. The designers expect a network manager to coordinate these items and provide them to OMI implementations in a cooperative, "friendly" collection of nodes. Another network protocol could, in principle, perform these functions as well.

Different implementations have different names for objects; OMI consistently uses the servers' names. The designers expect implementations to allow the network manager to enter servers' names into clients' nodes, where clients' names for objects are translated into servers' names.

Items that may need translation:

- *User identifier:* identifies the user in each request header (4.5.5, 5.3.1).

- *Group identifier:* identifies the user's group in each request header (4.5.5, 5.3.1).

- *Environment:* part of a global reference (5.3.3) denotes the namespace in which the accompanying gvn or nref appears. A client's environment should be translated into a server's name and environment (4.7, 5.3.3).

In addition, the agent and server exchange their names and passwords in the *connect* transaction (5.4.1). Their administration is the network manager's responsibility.

B.3 Authorization

Most database management systems provide authorization, the determination that the user has authority for the requested operation on a database. The system performs the function if authorized and returns an error to the application program if not.

A survey of MUMPS implementations revealed a wide variety of authorization methods. In most — but not all — only one user gets a user ID and several users share a group ID. Each database item is associated with a user ID and a group ID. (The association may be with the file that holds the globals, the globals themselves, or parts of globals). Different privileges for different operations then derive from matches of the user ID and group ID.

Most of the surveyed implementations perform authorization on the server node, where the necessary information is stored. At least one, however, performs all authorization on the client node by consulting a list of authorized globals for each user and sending only authorized requests to the server.

Finding no method subsuming both cases, the designers chose authorization on the server (described in 4.5.5), recognizing the additional implementation cost for those who presently perform authorization on the client.

Vendors may wish to let their servers recognize a "super-user" ID, given all privileges, for sessions with clients that send only authorized requests.

B.4 Protocol stack

The designers of OMI intend it to work well in implementations that include other network services, including proprietary protocols providing the same database functions. Because OMI relies on an externally supplied transport protocol, its success at sharing network hardware and software depends on the generality of the vendor's network software services.

OMI network managers face a more complex task than those purchasing a proprietary network from one vendor who takes responsibility for making everything work. As with any mixed-vendor system, the customer assumes that responsibility or pays a system integrator to do so. For a successful network the vendors must offer — and the customer must choose — a compatible stack of protocols at the transport level and at each lower level down to the hardware connection. (Gateways between unlike protocols are available.) Any method that provides an error-free, sequence-preserving path for messages containing all 8-bit characters is suitable.

NOTE - Therefore, a serial line interface could not use special characters for flow control or other purposes unless a protocol below OMI mediates to permit all 256 characters for OMI messages.

Transactions are isolated (see 4.3.3), but the *lock* operation (5.4.13) implies a lock manager that associates nrefs with the clients that have claimed them and denies other claims. The designers assume the lock manager to reside on the server's node. The OMI server relies on the lock manager to resolve *lock* requests; OMI simply transports them.

Implementers of client nodes should consider what to send to the lock manager when clients terminate normally or abnormally, and when disconnecting a session for different reasons.

Implementers of servers should address the matter of outstanding locked nrefs when a session terminates due to circuit failure. It may be prudent to give the network manager the choice of keeping them or dismissing the claims.

B.5 Sessions

The duration of a session is up to the implementer, considering the relative costs of establishing and maintaining the circuit that supports it. The OMI designers expect a session to last much longer than one transaction. It might last as long as both nodes are available, but it also might last for a single transaction only.

The agent process is part of the conception — and perhaps the implementation — of this standard (see 4.3.2). Most transport-level network products offer a limited number of virtual circuits, and some popular server implementations use rather expensive sessions, for example a subprocess per session. The agent allows many clients on a node to share a session with a server. The agent, as the caller of the underlying transport level, can maintain the session while clients come and go.

By allowing the multiplexing of requests, OMI gains more efficient use of network resources at the cost of identifying the user in each transaction.

B.6 Complex operations

B.6.1 Lock

4.3.4 describes a complex lock, in which a client requests an atomic lock involving nrefs on different servers. The agent makes single *lock* requests to the servers and, if some of them fail, unlocks those that succeeded and then tries the whole procedure again.

This method avoids deadlock between conflicting complex locks, but it can generate much network traffic as agents repeat their requests, and (given just the wrong timings) a condition known as live-lock. Therefore, the agent should introduce a delay before retrying the lock. If the operation continues to fail, the length of the delay should be increased. While this "backing off" does not guarantee avoidance of live-lock, it greatly reduces the probability. The initial delay time, increase in delay and maximum delay are left to the implementer.

OMI includes requests to unlock all nrefs held by one client, and to unlock all nrefs held by all clients on a node. These requests facilitate the argumentless LOCK and simplify the agent, which would otherwise have to remember what nrefs it holds on the server's node.

B.6.2 Merge

While future versions of OMI may support the merge operation (see 4.1) directly, the present standard provides the tools for an agent to accomplish it. To merge from client to server, send *set* requests in the same sequence as in a local merge. To merge from server to client, walk the server's tree with *order* requests or *query* requests while making *get* requests for the values.

B.7 Registering implementations

No registration of an implementation of this standard is necessary, but the MUMPS Development Committee maintains a registry of certain elements for those who wish to use it:

- *Extensions:* (4.11, 5.4.1) afford increased performance, new functions, and testing of possible features for new versions. Extensions, by definition not part of this standard, must be registered.

- *Implementation identification:* (5.4.1) may be the empty string without registration, but registration permits related implementations to recognize each other when establishing a session, perhaps to agree on untranslated characters or certain extensions, and to aid the network manager.

B.8 Compliance verification

When two vendors' systems with allegedly compatible stacks of protocols cannot communicate, who is right and who is wrong? Ideally, a disinterested authority would provide a verification service for all the protocols. In reality there is no formal verification process for this standard. The lack of such a process has not prevented successful installations of other protocols.

Verification by consensus among vendors is an important consideration in the design of OMI, influencing its reliance on existing lower-level protocols, its simplicity, and its incorporation of detailed and explicit error messages.

The M Technology Association sponsors OMI implementers' workshops at its annual meetings to promote the implementation and testing of the protocol. While quite unofficial, the workshops play an important role in establishing implementation guidelines. In the future, this group may publish formal recommendations for layering OMI on various protocols, and it could develop a test suite to validate OMI compliance.

The MUMPS Standards Interpretation Review Board, mentioned in the foreword, reviews and arbitrates disputed interpretations of this Standard, although it does not validate specific implementations.

Annex C

(Informative)

Extensions

Plans for future versions of this standard, and proposed extensions to the ANSI/MDC X11.1 standard defining the MUMPS language should be of interest to those wishing to implement, use, or extend this standard. The MUMPS Development Committee welcomes participation by all who want to contribute.

C.1 Requests and responses

The MUMPS Development Committee has reserved certain operation types, shown in table C.1, for future versions.

C.1.1 Increment

(Operation type 14) This operation increments the value of a global variable.

The fields of the *increment* request and their sequence shall be:

a *Replicate flag:* <SI> indicates whether the server should replicate the operation (see 4.6). 0 means no, 1 means yes. All other values are reserved.

NOTE - Servers are not required to support replication. Those that do support it shall honor this field.

b *Global reference:* <LS> the global reference that is to be incremented.

c *Increment value:* <SS> the numeric value by which to increment the global variable.

The one field of the *increment* response shall be a long string <LS>, the value of the global variable after it has been incremented.

C.1.2 Reverse query

(Operation type 26) The *reverse query* operation fetches the entire global reference preceding the global reference <LS> of its request. The final subscript of the requested global reference may be the empty string.

The *reverse query* response shall comprise 1 field, a long string <LS> containing the preceding global reference. If there is no preceding global reference, this field shall equal the empty string.

C.2 Other database support

Some less well defined operations are under consideration to support present and planned database functions:

- *Merge:* To merge a server's tree of global variables with a client's, an initiating and continuation request may be defined (see B.6.2). Transferring a potentially unlimited number of variables, this function may require the agent and server to retain some information across transactions.

```
+---------------------------------------------------------+
¦  <SI>     ¦      <LS>           ¦      <SS>             ¦
¦ Replicate ¦ <LI> ¦ Global Ref ¦ <SI> ¦ Increment Value ¦
+---------------------------------------------------------+
```

Figure C.1 - Increment request

```
+-------------------------+
¦       <LS>              ¦
¦  <LI> ¦ Global Value    ¦
+-------------------------+
```

Figure C.3 - Increment response

```
+------------------------------+
¦       <LS>                   ¦
¦  <LI> ¦ Global Reference     ¦
+------------------------------+
```

Figure C.5 - Reverse query request

```
+--------------------------+
¦      <LS>                ¦
¦ <LI> ¦ Global Reference  ¦
+--------------------------+
```

Figure C.7 - Reverse query response

Table C.1 - Reserved operation types

Type	Name	Description
14	Increment	Increments the value of a global variable.
26	Reverse query	Requests the previous gvn in reverse collating order. Returns the empty string if no further gvns are defined.

- *Combined order and get:* Operations similar to order, query, and their reverse forms would return the value of the next global variable in addition to its subscript or gvn.

- *Transaction processing:* Support for transaction start, commit, and rollback will require careful division of functions between OMI and the application level. Distributed transaction processing is expected to require many changes to the protocol itself.

- *Portability limits:* Proposed expansion of several MUMPS language elements for portable programs will have little effect on the standard, but may require implementation changes.

C.3 Beyond database functions

MUMPS users have often expressed needs for process-to-process communication, remote device support, remote job entry, remote program loading, and other functions available from proprietary protocols. OMI may be an appropriate vehi

cle for some of these. Its designers will have to balance the virtues of simplicity and functionality.

C.4 Performance enhancements

The *connect* operation (5.4.1) now allows only one request outstanding in a session. Multiple requests are anticipated, necessitating more complex error reporting and recovery mechanisms.

The agent could pack many requests into a message and the server could pack many responses, thus spreading the overhead of message transmission among many transactions. This, too, would require more complex error reporting and recovery.

C.5 Other protocols

The evolution of related protocols may prompt the division of OMI into session-layer and presentation-layer protocols. Some of its functions might then be transferred to other protocols at these levels. The MUMPS Development Committee intends to move this standard toward compliance with OSI standards as they emerge.