

IEEE Standard for Media Management System (MMS) Media Management Protocol (MMP)

Sponsor

**Storage Systems Standards Committee
of the
IEEE Computer Society**

Approved 21 June 2000

IEEE-SA Standards Board

Abstract: All relevant behavior and states of the Media Management System (MMS) and the Media Manager (MM) from the time the MMS application successfully establishes a session with the MM, until the session is severed is described. This standard specifies the behavior of the MM itself in response to certain MMP commands and events that occur during the existence of the MM, including but not limited to booting the MM and the un-commanded exit of a MMS client.

Keywords: application independent, architecture, automated tape library, content neutral, data model, device manager, distributed, drive manager, fully scalable, heterogeneous environment, information access, interoperability, interoperable, language neutral, library management, library manager, media management, media neutral, middleware, opensource, operating system independent, platform-independent, protocol based, removable media, robotic tape library, secure, software, storage, storage management, storage system, system architecture

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 15 December 2000. Printed in the United States of America.

Print: ISBN 0-7381-2510-5 SH94864
PDF: ISBN 0-7381-2511-3 SS94864

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

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

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

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

[This introduction is not part of the IEEE Std 1244.3-2000, IEEE Standard for Media Management System (MMS) Media Management Protocol (MMP).]

This is one of five MMS standards balloted together under IEEE Project 1244 for Storage System Design:

IEEE Std 1244.1-2000—IEEE Standard for Media Management System (MMS) Architecture

IEEE Std 1244.2-2000—IEEE Standard for Media Management System (MMS) Session Security, Authentication, Initialization Protocol (SSAIP)

IEEE Std 1244.3-2000—IEEE Standard for Media Management System (MMS) Media Management Protocol (MMP)

IEEE Std 1244.4-2000—IEEE Standard for Media Management System (MMS) Drive Management Protocol (DMP)

IEEE Std 1244.5-2000—IEEE Standard for Media Management System (MMS) Library Management Protocol (LMP)

This MMS MMP standard (IEEE Std 1244.3-2000) is best understood in the context of the others by reading the first two sections of the Architecture (IEEE Std 1244.1-2000). These five standards are the first in a suite of ten approved projects for the IEEE MMS, and all ten might well have been a single standard. However, their separation was maintained to make conformance easier and individual evolution possible.

By permission of SGI™ this standard is based on the SGI™ OpenVault™ product, which SGI™ has released to the open source community. However, the IEEE Media Management System, developed by the IEEE Storage System Standards Working Group (SSSWG), is a significant refinement and extension of OpenVault™. The IEEE Mass Storage System Reference Model (MSSRM) influenced OpenVault™ design before work began on the MMS, and the present OpenVault™ reflects strong influence in its development by the IEEE SSSWG.

At the time this standard was completed, the Storage System Standards Working Group (SSSWG) had the following members:

John L. (Jack) Cole, *Chair*

Curtis Anderson
Samuel S. Coleman*
R. Troy Eberlein
Bruce K. Haddon
T. Dixon Hutchinson

Merritt E. Jones*
Jan Klier
Stuart Kreitman
Paul Lockwood

John Merrill
Geoffrey G. Peck
Alan Rollow
Murali Sathyanarana
Joel N. Williams*

(* Denotes a charter member of the SSSWG)

Technical editors of these five standards are as follows:

IEEE Std 1244.1-2000—IEEE Standard for Media Management System (MMS) Architecture

—Geoffrey G. Peck (Primary Editor for Architecture)

—Curtis Anderson

—Murali Sathyanarayana (Primary Editor for the Data Model)

—Curtis Anderson, Joel N. Williams, T. Dixon Hutchinson, Alan Rollow

More information on OpenVault™ is available at <http://www.sswg.org>.

IEEE Std 1244.2-2000—IEEE Standard for Media Management System (MMS) Session Security, Authentication, Initialization Protocol (SSAIP)

—Bruce K. Haddon (Primary Editor, 2000–present)

—Jan Klier (Primary Editor, 1998-2000)

—Curtis Anderson, Joel N. Williams

IEEE Std 1244.3-2000—IEEE Standard for Media Management System (MMS) Media Management Protocol (MMP)

—Murali Sathyanarayana (Primary Editor)

—Curtis Anderson, Joel N. Williams, Dixon Hutchinson, Alan Rollow

IEEE Std 1244.4-2000—IEEE Standard for Media Management System (MMS) Drive Management Protocol (DMP)

—Joel N. Williams (Primary Editor)

—Murali Sathyanarayana

IEEE Std 1244.5-2000—IEEE Standard for Media Management System (MMS) Library Management Protocol (LMP)

—Joel N. Williams (Primary Editor)

—Murali Sathyanarayana

Sponsor chairs of this standard are as follows:

—Merritt E. Jones (1993-2000)

—John L. (Jack) Cole (2000)

The following members of the balloting committee voted on this standard:

Curtis Anderson
Brian A. Berg
Paul Buerger
John L. Cole
George B. Cole
Don Doerner
Richard T. Eberlein
Robert J. Gagliano
Mark R. Gary

Michael V. Hardy
Ronald W. Huff
T. Dixon Hutchinson
Merritt Jones
Michael P. Kearney
Linda Kempster
Charles M. Kennedy
Ben Kobler

Stuart K. Kreitman
Paul Lockwood
Geoffrey Peck
Alan R. Rollow
Murali Sathyanarayana
Yoshitake Shinkai
Bob Snead
Robert Snively
Joel N. Williams

When the IEEE-SA Standards Board approved this standard on 21 June 2000, it had the following membership:

Donald N. Heirman, *Chair*

James T. Carlo, *Vice Chair*

Judith Gorman, *Secretary*

Satish K. Aggarwal
Mark D. Bowman
Gary R. Engmann
Harold E. Epstein
H. Landis Floyd
Jay Forster*
Howard M. Frazier
Ruben D. Garzon

James H. Gurney
Richard J. Holleman
Lowell G. Johnson
Robert J. Kennelly
Joseph L. Koepfinger*
Peter H. Lips
L. Bruce McClung
Daleep C. Mohla

James W. Moore
Robert F. Munzner
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Akio Tojo
Donald W. Zipse

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*

Donald R. Volzka, *TAB Representative*

Catherine K. N. Berger
IEEE Standards Project Editor

AMPEX is a registered trademark of AMPEX Corporation.
DDS is a registered trademark of Sony Corporation.
DTF is a registered trademark of Sony Electronics, Inc.
DLT is a registered trademark of Quantum Corporation.
DST is a registered trademark of AMPEX Systems Corporation
Exabyte is a registered trademark of Exabyte Corporation.
IBM is a registered trademark of the International Business Machines Corporation.
Magstar is a registered trademark of International Business Machines Corporation.
Quantum DLT is a registered trademark of Quantum Corporation.
SGI, Silicon Graphics, OpenVault, and IRIS FailSafe are trademarks of Silicon Graphics Incorporated.
STK is a trademark of Storage Technology Corporation.
UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.
Windows NT is a registered trademark of Microsoft Corporation.

All other tradenames and trademarks in this standard are those of their respective owners.

Contents

1.	Overview	1
1.1	Scope	2
1.2	Purpose	2
1.3	Conformance	3
1.4	Rationale	3
2.	References, definitions	4
3.	MMP Operational model	4
3.1	Objects and their attributes	4
3.2	Relationships between objects	4
3.3	Persistent storage	5
3.4	Command set	5
3.5	Command fault recovery	6
3.6	Privilege model	7
3.7	MMP Protocol overview	7
3.8	MMS Startup and shutdown	9
4.	Operation of commands	10
4.1	Evaluation of commands	10
4.2	MMP Responses and command sequencing	13
4.3	Error model	15
5.	Semantics of common syntactic elements	19
5.1	Syntactic element name: OBJECT."attributename"	19
5.2	Clause name: volname	20
5.3	Clause name: match	21
5.4	Clause name: order	23
5.5	Clause name: number	24
5.6	Clause names: report, reportmode, cpreport, and cpreportmode	26
5.7	Clause name: text	29
5.8	Clause name: reqid	29
5.9	Glossary of match subfunctions	30
5.10	Treatment of invalid operands	31
5.11	Numeric and time values	33
6.	MMP command descriptions	33
6.1	Media management	34
6.2	Session management	55
6.3	Taskmanagement	60
6.4	System management	62
6.5	Data management	64
6.6	Device management	76
6.7	Operator interface	86
7.	Error responses	101

IEEE Standard for Media Management System (MMS) Media Management Protocol (MMP)

1. Overview

The IEEE MMS Media Management Protocol (MMP) is the language that Media Management System (MMS) client and administrative programs use to communicate with the Media Manager (MM).

At the heart of the MMS is the MM, which provides mechanisms to manage and facilitate access to removable media (resident in robotic or non-robotic libraries). The MM interoperates with other components of the MMS to allow the following:

- a) Application vendors to easily and quickly implement uniform platform-independent solutions that access removable media.
- b) Easy implementation of interoperable and portable solutions that share and manage removable media, while eliminating the need to write custom solutions.
- c) The MM manages media and provides access to media when requested. The MMP language provides rich capabilities, including functionality in the following areas:
 - *Media Management* functions for principals (applications) of the MMS to access and manage removable media.
 - *Session Management* functions for applications to control the session.
 - *Task Management* functions for applications to control the system and the environment in which they operate.
 - *System Management* functions for controlling the operation of the MM.
 - *Data Management* functions for implementing administrative-level capabilities, and for applications to view and modify meta-data associated with the media.
 - *Device Management* functions for the control of devices.
 - *Operator Interface* functions for operations personnel to perform routine tasks.

The MMP is an application-level, text-oriented interface protocol running over TCP/IP. ISO-10646 characters, encoded in UTF-8, are used as the character set. This protocol is an asynchronous, bi-directional protocol.

Applications that utilize the MMP are afforded varying levels of privilege, which define the set of operations it may perform. This version of the standard defines two levels: standard and administrator privilege levels, with administrator being a proper superset of the standard level. This standard uses the terms non-privileged and privileged to mean standard and administrator, respectively. Future versions may include additional privilege levels.

Client access to the MM is session-oriented. See IEEE Std 1244.2-2000 for more detailed information on session initiation. After session initiation, the application may send commands to the MM to query and modify various pieces of system state information or to exercise physical device control and request access to media.

1.1 Scope

The IEEE Media Management Protocol (MMP) is used when an MMS application requests the services of the MM, including the following:

- a) Access to media
- b) Device management functions
- c) Routine operational functions
- d) MMS administration

This protocol specifies the syntax and the semantics associated with various commands and protocol messages that pass between an MMS application and the MM. In addition, this standard defines certain aspects of the operation of the MM itself in response to various events that occur in the MMS, for example, the initiation and termination of various components of the MMS.

Since this protocol is only used in the context of the MMS, this standard cannot be understood without a thorough understanding of its architecture and the underlying Data Model as described in IEEE Std 1244.1-2000.

This standard does not dictate or favor any particular implementation scheme for the MM. It is a standard for those interested in implementing applications that utilize the services of the MM. For example, the protocol requires that the MM employ a persistent store to keep track of various pieces of information. However, this specification does not imply any particular structure of the store. An MM implementation shall record errors and conditions that occur during the operation of the MMS. To achieve this, the Data Model defines a specific table, called the MESSAGE table. An MM implementation could provide additional logging capabilities that are beyond those defined by this standard.

1.2 Purpose

The purpose of this standard is to specify the syntax and semantics of a language, the MMP. This language conveys commands between an MMS application and the MM after an application has successfully negotiated a connection through the Session Security, Authentication, and Initialization Protocol (SSAIP), IEEE Std 1244.2-2000. This allows any IEEE Std 1244.3-2000 compliant MM and any IEEE Std 1244.3-2000 compliant MMS application to communicate as described in this standard.

This standard describes all relevant behavior and states of the MMS and the MM from the time the MMS application successfully establishes a session with the MM, until the session is severed.

In addition, this standard specifies the behavior of the MM itself in response to certain MMP commands and events that occur during the existence of the MM, including but not limited to booting the MM and the uncommanded exit of an MMS client.

1.3 Conformance

An MM conforms to this standard if it can send and receive protocol messages in the syntax described in this standard, acts in accordance with the semantics of those messages, and operates in the fashion described in this standard. An MM shall also conform to the abstractions defined in the Data Model clause of IEEE Std 1244.1-2000.

This standard specifies version 1.0 of the MMP. It is expected that this protocol will be revised periodically, at which time addenda to or revisions of this standard will be issued. MMS clients and the MM negotiate for a mutually agreeable version of the MMP using SSAIP (IEEE Std 1244.2-2000).

In order to ensure interoperability between applications and the MMS, and between components that comprise an installation of the MMS, it is required that MMS components adhere to this standard.

When implementing MMP in an MMS component, the full protocol for a given version of MMP shall be implemented. Supersetting or subsetting is not permitted. Multiple versions of MMP may be implemented within a given component. All versions implemented shall fully conform to their respective MMP standard; additions, modifications, and extensions are not permitted.

All components shall comply with the relevant clauses of the Data Model as defined in IEEE Std 1244.1-2000.

This standard defines consistent methods by which an application and the MM interact, and methods by which an application may control actions by drive and library managers. Customization of the basic functions of the MMS is permitted in two ways: through user-defined attributes and through commands (**request**, **release**, **respond**, **cpattribute**, **cpshow**, **cpscan**, and **cpreset**), which allow direct interaction with devices.

A non-conforming superset of the MMP might be used to experiment with new features, which at some future date might be incorporated into the standard. If such an experimental version of the MMP is implemented, the language name shall be changed. It is suggested that such non-conforming versions be supersets of an existing, defined version of this language.

To conform to the MMS as a whole, a system shall incorporate all the basic software components described in IEEE Std 1244.1-2000. These components, an MM, Library Managers (LMs), and Drive Managers (DMs) shall have the functionality described in the associated standards (IEEE Std 1244.1-2000–IEEE Std 1244.5-2000). It is permissible to create individual components that comply with one or more of the language standards but do not fit into a full MMS; this is considered a partially conforming MMS. (For example, a reduced-function MM that fully supports MMP may incorporate internal implementations of an LM and a DM. This is considered partial conformance, because the MM cannot accommodate additional LMs and DMs since it does not implement LMP (IEEE Std 1244.4-2000) and DMP (IEEE Std 1244.5-2000). However, a fully conforming MM shall be capable of completely supporting both LMP and DMP.

1.4 Rationale

The primary motivation for the MMP is to provide applications that use removable media with a simple and uniform, platform-independent means of accessing, sharing and managing media, while alleviating the need to develop expensive, proprietary solutions. The resulting protocol and the abstractions provided therein allow for easy management and sharing of a diverse range of media and distributed devices in a coherent and scalable fashion.

The MM provides applications with the ability to allocate and mount media, store meta-data, and manage libraries and drives in a platform-independent fashion. An application need only be implemented once to the interface, and does not have to worry about interoperability and data interchange issues, nor is the application limited to a set of devices/media.

2. References, definitions

A suite of IEEE Std 1244 standards describes the MMS. This modular approach to standardizing an MMS permits a granularity of conformance not possible otherwise, and allows a degree of independent evolution for the components of an IEEE MMS. Nevertheless, the overall architecture of the IEEE MMS is described in IEEE Std 1244.1-2000, MMS Architecture, and fundamental information (including the data model) that is common to all related standards is communicated through that standard. Reference to the MMS Architecture is crucial in understanding the MMP.

This standard shall be used in conjunction with the following standards. When the following standards are superseded by an approved revision, the revision shall apply.

IEEE Std 1244.1-2000, IEEE Standard of Media Management System (MMS) Architecture.¹

IEEE Std 1244.2-2000, IEEE Standard for Media Management System (MMS) Session Security, Authentication, Initialization Protocol (SSAIP).

IEEE Std 1244.4-2000, IEEE Standard for Media Management System (MMS) Drive Management Protocol (DMP).

IEEE Std 1244.5-2000, IEEE Standard for Media Management System (MMS) Library Management Protocol (LMP).

ISO/IEC 10646-1:1999, Information Technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane.²

Additional references common to all IEEE MMS standards can be found in Clause 2 of IEEE Std 1244.1-2000; definitions can be found in Clause 3 of IEEE Std 1244.1-2000.

3. MMP Operational model

3.1 Objects and their attributes

The MMS defines more than 40 types of objects that make up a media environment. These objects embody the abstractions of the various elements of an MMS. A complete description of the objects, their attributes, and how they relate to each other can be found in IEEE Std 1244.1-2000.

3.2 Relationships between objects

The objects defined in the MMS are all related to each other, and each is grouped into one of several clusters. Relationships within a cluster are mostly containment. Relationships between clusters are designed to induce intuitive semantics when examining object combinations that include objects from multiple clusters. Some relationships are physical, for example partitions, sides, and cartridges, or slots, bays, and libraries. And some relationships are logical, for example, applications, volumes, and partitions.

These relationships shall be understood to effectively implement/utilize the MMP.

¹IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (<http://standards.ieee.org/>).

²ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

3.3 Persistent storage

The MM stores persistent information embodied in the objects described in the Data Model. This includes the following types of information:

- a) Description information about media.
- b) Information about applications, LMs, and DMs.
- c) Configuration information reflected in the Data Model.
- d) Statistical information, specifically information on cartridge-drive usage.
- e) The last known configuration information from LMs and DMs.
- f) Attributes created by applications, LMs, and DMs.

In this version of the protocol, transient information such as commands in progress, client sessions, and LM or DM requests may be discarded when the MM is restarted.

An MM implementation may employ any storage mechanism that it desires, ranging from simple flat files to complex databases. This standard does not dictate, nor does it favor, any specific underlying storage architecture.

This standard provides some guarantees against data loss in the face of a crash. The storage mechanism shall allow the MM to maintain a logical consistency of the data even under such conditions. For example, a client requests allocation of a volume. The allocation requires a new **VOLUME** record to be created and the *CartridgeNumberVolumes* attribute to be incremented. Suppose the MM crashes while processing the **allocate** command. When the MM is restarted, one of the following shall be true:

- The **VOLUME** record has been created and the *CartridgeNumberVolumes* attribute has been incremented.
- The **VOLUME** record has not been created and the *CartridgeNumberVolumes* attribute has not been incremented.

3.4 Command set

MMP supports a rich range of commands. These commands fall into several different categories. Each command may be described as either data model-oriented or operation-oriented, depending on the nature of the actions that are involved when executing the command.

3.4.1 Type based view of the command set

Data Model commands: These commands perform a very specific function involving the Data Model. These commands do not imply a direct involvement of LMs and DMs. For example, **attribute** allows a client to modify attributes on an object in the Data Model. The following commands are Data Model oriented commands: **allocate**, **attribute**, **cancel**, **create**, **deallocate**, **delete**, **goodbye**, **locale**, **privilege**, **rename**, and **show**.

Operation commands: In general, these are commands that provide access to media for administrators and operators (via human interface applications), or client applications. The commands encompass a broad set of functions that affect more than just the Data Model objects instantiated in the MM. These commands include: **accept**, **begin-end**, **cattribute**, **cpscan**, **cpshow**, **cpreset**, **eject**, **inject**, **mount**, **move**, **release**, **respond**, **shutdown**, and **unmount**. Execution of these commands includes examining attributes of various objects in the Data Model in many cases. These attributes influence the outcome of the command. For

example, the *DriveGroupName* attribute of a drive influences whether a drive is included as a candidate for a non-privileged mount command.

3.4.2 Function based view of the command set

The same MMP commands might also be grouped into the following categories based on the area of functionality that each command deals with:

Media management: These commands deal with access to media. **mount** allows a client to access data on media. **unmount** lets a client relinquish access to a mounted cartridge. **allocate** lets a client gain ownership of un-owned portions of media (volumes). **deallocate** allows a client release ownership of a volume. Clients may change the name of a volume using **rename**.

Session management: The commands in this group deal with management of the operational characteristics of session and the MMS services. **goodbye** terminates a client session. **locale** allows a client to modify the operating locale of the current session. A client can modify the privilege level of the current session using **privilege**. **shutdown** causes the MMS to terminate and optionally restart its services.

Task management: **cancel** allows cancellation of blocked tasks such as mounts.

System management: **shutdown** allows a privileged client to terminate the MMS.

Data management: These commands allow clients of the MMS to view and modify various objects and attributes in the Data Model. **attribute** modifies attributes of objects in the Data Model. **show** allows an MMS client application to view attributes of various objects. **create** and **delete** create and destroy a single instance of an object, respectively.

Device management: These commands allow an MMS client application to interact directly with an LM or DM. **cpattribute** modifies attributes in an LM or DM. **cpshow** displays information about LM and DM attributes. **cpscan** directs an LM to scan portions of a library. **cpreset** causes an LM or DM to disconnect from the managed device and reinitialize its connection to the MM.

Operator interface: These commands perform routine tasks involving libraries and drives. **move** allows cartridges to be moved from one location in a library to another. **inject** and **eject** allow insertion and ejection of cartridges to and from a library. **accept**, **release**, and **respond** allow an operator to deal with requests posted by an LM or DM.

Commands that involve physical operations on media in an LM or DM are limited to one unit of media per command. Similarly, commands that cause a change in the ready state of an LM or DM, that cause a change in the state of attributes in an LM or DM, or that directly cause physical operations to be performed in an LM/DM are also limited to one LM/DM per command. **begin-end** allow grouping of **mount** and **unmount**.

3.5 Command fault recovery

This standard allows an MM implementation to define retry strategies in the cases where a command encounters errors while it is executing and retries are possible. For example, an MM may retry a command that has already been dispatched to an LM, but failed as a result of a transient error. In other cases, the standard dictates that the MM will either complete all actions involved in a command or respond with an error without performing any of the actions. The attribute command, for example, exhibits this behavior.

3.6 Privilege model

Security of communications between MMS components is provided by SSAIP (IEEE Std 1244.2-2000). In this version of the MMP standard, data within the MMS is protected through two security levels: privileged and non-privileged. When a client initiates a session with the MM through SSAIP, it is recognized/authorized as either a privileged or non-privileged client. A non-privileged client only has access to the non-privileged subset of the MMP language. This subset is subject to the following restrictions:

- a) The non-privileged client has no visibility into other client's namespaces for volumes.
- b) The non-privileged client has read-only access to drive and library attributes.
- c) The non-privileged client has no direct ability to create or destroy objects.

An administrative client has access to the full power of the MMP language, implying read-write access to all applicable attributes on any object in the system, and the ability to create and destroy objects.

Future versions of MMP may define more than two levels of security.

3.7 MMP Protocol overview

The MM and MMS clients communicate over TCP connections that are established using SSAIP (IEEE Std 1244.2-2000). At the time an MMS connection is established, SSAIP provides identification, and optionally, authentication of the MMS client. Successful completion of the Session Initialization Phase in SSAIP results in an active MMP session.

The SSAIP is also employed throughout the duration of an MMP session, i.e., MMP operates on top of a layer of SSAIP. SSAIP provides security functions while an MMP session is in progress. Refer to IEEE Std 1244.2-2000 for a discussion of session initialization.

The MMP language is a text-oriented language and is human-readable. It uses the ISO-10646 character set encoded in UTF-8.

The MMP uses a combination of objects, attributes on objects and commands to accomplish various tasks that are required in a media-management system. The MMP language consists of a collection of commands that are to be executed subject to certain constraints that are specified along with the command. The MM evaluates the constraints and executes the command on the set of objects that satisfy those constraints. The Data Model defines a set of named objects. Each object includes a set of system-attributes. Some objects may allow user-specified attributes to be associated with them. The Data Model defines relationships between the various different objects. The Data Model also defines a set of permissions (currently a set of one element, either privileged or non-privileged) for each attribute/object, in addition to the domain of values that may be assigned to an attribute. The native data type supported by the Data Model is a string. Some system-defined attributes represent time-stamps, Boolean, numeric, or enumerated values, but are stored as strings.

The MMP command set allows a user to use objects and attributes in various commands in the following manner. The user may

- a) View information on named attributes of objects.
- b) View information on all attributes of a named object, without requiring enumeration of the attribute names.
- c) Name attributes in order to modify them.
- d) Specify arbitrary expressions that involve attributes.

The values for some attributes influence the outcome of a command. The descriptions of the commands list the use of such attributes.

This standard defines a set of privilege levels, as described in 3.6. At any given instant, an MMP session has exactly one privilege level associated with it. Each command in the protocol is executed in the context of the limitations the current privilege level dictates. For example, a *non-privileged* client may only mount on drives in the drivegroups that it is allowed access to, unlike a *privileged* client that may mount cartridges on any drives in the MMS (assuming physical accessibility). Similarly, this standard also allows a set of locales. Each command is executed in the context of the current locale.

Some commands issued during an MMP session cause changes in the MMS that persist beyond the duration of a session. For example, creation of a user-defined attribute on a volume remains until either the attribute is explicitly destroyed or until the volume is deleted. Future versions of MMP may allow a client to disassociate from a session and reattach to it at a later time while maintaining transient state across reattachment of sessions.

The MMP command-response format is semi-asynchronous. After submitting each command, the client shall wait for the MM to acknowledge the receipt of the command, but need not wait for the results of the command before sending the next one. A successful acknowledgement of a command implies that the MM successfully parsed and queued the command. After the initial acknowledgement, the MM generates a single response to each command. Refer to IEEE Std 1244.1-2000 for command throttling.

Each MMP command includes a task identifier (TID). While the format of the TID is arbitrary, clients should create TIDs that are unique within the context of a connection to the MM. All responses to an MMP command will include the TID of the initiating command. This enables the client to associate responses with commands that it has issued. If the client can guarantee that uniqueness of task TIDs within the context of a session, then the client can determine which response goes with which command.

The following examples with real MMP commands may serve to clarify this idea:

The client sends the following command to the MM:

```
mount task ["1"] volname ["v1"] ;
```

The MM sends an acknowledgement of receipt:

```
response task["1"] accepted;
```

Sometime later, the MM sends back a completion status for the command:

```
response task["1"] success ;
```

Since the client can determine which response came from the execution of each individual command, a more sophisticated sequence can occur in which a new command is sent to the MM before the MM responds with the completion status of the first command.

```
mount task ["1"] volname ["v1"] ;
```

```
response task["1"] accepted;
```

```
attribute task ["2"] volname ["v1"] set [VOLUME."Color" "green"] ;
```

```
response task ["2"] accepted;
```

```
response task ["2"] success;
```

```
response task ["1"] success;
```

Note that in this example, the client sent its second command before the first command had completed execution. In fact, in this particular example, the second command completed its execution before the first.

Refer to IEEE Std 1244.1-2000 for a description of command sequencing and how to achieve command throttling.

3.8 MMS Startup and shutdown

This subclause examines various startup, shutdown, and failure events that could occur within an MMS. A general overview of how the MMS reacts in response to such events is provided. Such events could occur under various situations, some of which require special handling. This standard does not attempt to do an exhaustive analysis for each event-state combination. While this standard acknowledges that multiple failure events could occur, this standard deals with single failure cases.

The languages of the MMS are implemented on top of TCP/IP and are subject to the limitations of these protocols. For example, it is not possible for the MM to distinguish between a client system crash versus a network partition.

It is possible for some implementations of the MM to be able to deduce conditions of a host failure.

3.8.1 Application events

Initiation of an application session is covered in detail in IEEE Std 1244.2-2000.

A session between a client and the MM may be terminated in the following two ways.

- a) Normally, by the execution of goodbye. In this case the MM will process pending and new commands as specified in goodbye.
- b) Abnormally, by the closure of the network connection between the MM and the application. In this case the MM will:
 - 1) Release any accepted requests.
 - 2) If the application had any cartridges mounted, then the MM may reclaim these resources. However, it is possible for other components of the application to be accessing the media at the time of the failure. The **detach** DMP command allows an MM to test whether an access handle is in use.
 - 3) If **detach** returns with an indication that all handles have been deleted, then the MM shall destroy all state associated with the session.

3.8.2 MM Events

A client may discover that its session with the MM has been prematurely terminated due to network partition or other system failures. An application may periodically attempt to re-establish a connection to the MM, however, state information from the previous session will be lost. Applications will have to re-issue commands for which responses had not yet been received. It is possible that pending commands have completed but responses to them have not arrived at the application. It is the application's responsibility to determine, upon re-establishing the session, the state of completion of these commands. Note that the tasks issued from a prior application session are no longer visible in a new incarnation of the MM.

When a session between an LM or DM and the MM is prematurely terminated, the LMP or DMP commands may be aborted, resulting in a failure response to the MMP client. The client in these cases may re-issue the command. The LM or DM should be trying to re-establish its connection with the MM.

4. Operation of commands

4.1 Evaluation of commands

The MM implements each command by identifying a working set of one or more objects to operate upon, and then performing sequences of operations on the identified set of objects and invoking commands to other MMS components as necessary.

The construction of the working set involves the application of different sets of constraints on the universe of all objects. These constraints include the following:

- a) *Implicit* constraints
- b) *Explicit* constraints
- c) *Derived* constraints

The description of each command enumerates the set of implicit constraints imposed by that command.

4.1.1 Implicit constraints

Implicit constraints can be further divided into the following:

- a) Constraints that dictate which target objects are involved.
- b) Constraints that are command-specific.
- c) Constraints that are privilege-based.

4.1.1.1 Target Objects

Each command determines the set of target objects to be included in the working set. For example, **unmount** will include drive objects into the working set. The MM always identifies a basic set of target objects that are included in the working set of a command. The eventual working set may include more objects than just target objects, since the **match** clause allows the application great freedom. Table 1 lists the base set of target objects associated with each command:

4.1.1.2 Command-specific constraints

These constraints generally dictate eligibility of target objects. Each command implies a set of constraints on the target objects that shall be met in order to decide inclusion or exclusion of the target object in the working set. For example, **cpattribute** sent to an LM will include only eligible LMs. One eligibility criterion is that an LM shall be currently connected to the MM.

4.1.1.3 Privilege-based constraints

The MM takes into account the current privilege level of the application issuing the command. Each privilege level dictates a set of constraints for a given command. For example, an **allocate** command issued by a standard privilege level application will exclude partitions on cartridges owned by other applications, and will apply only to those cartridges in the appropriate CartridgeGroup.

Table 1—Target objects associated with each command

Command	Target objects
mount	Cartridge/drive
unmount	Drive
deallocate	Volume
rename	Volume
create	Object specified in type clause
delete	Object specified in type clause
attribute	Object(s) specified in the set and unset clauses that are visible to the application
show	Any visible to the application
cpscan	LM
cpattribute	LM or DM
cpshow	LM or DM
cpreset	LM or DM
shutdown	MMS
cancel	Task
privilege	Current session
locale	Current session
goodbye	Current session
accept	LM or DM request
respond	LM or DM request
move	Side
eject	Cartridge
inject	Library
allocate	Partition
release	LM or DM request
begin	Current session
end	Current session

4.1.2 Explicit constraints

MMP is a declarative language. An application may explicitly declare criteria that shall hold true in order for the MM to find a solution for the command. There are many clauses in the various MMP commands where an application may express constraints, ranging from complex expressions to trivial constraints. For example **match** allows constraint expressions of arbitrary complexity, while **where** specifies a very simple constraint. **match**, **order**, and **number** are allowed in commands whose working sets have the potential for more than one instance of an object or the potential for more than one object. For example, **locale**, **goodbye**, and **privilege** all refer to the current session and hence do not require **match**, **order**, and **number**.

4.1.3 Derived constraints

The application of derived constraints is not obvious from the syntax of the MMP. Recall that the MMP allows an unrestricted constraint expression-based specification for many commands. The syntactic elements included in the language allow for expressions constructed using objects, attributes, and values. Since any arbitrary expression involving arbitrary tables may be employed in the context of most commands, the MM is responsible for arriving at the set of target objects, given the context of the command and the objects referenced in the command. Based on the objects referenced in the command and the command being executed, the MM applies a set of derived constraints to arrive at the combinations of target objects. Objects are classified into clusters. A handful of constraints define relationships between clusters. Relationships within a cluster are mostly ones of pure containment. The complete set of derived constraints is listed in 9.2 of IEEE Std 1244.1-2000 and is designed to be a small but intuitive set. The following clauses, when specified and required, are all used to determine the set of derived constraints imposed on the command: **match**, **volname**, **reqid**, **order**, **number**, **set**, **unset**, and **report**.

For example, the command:

```
show task["1"] match [streq (DRIVE."color" "green")]

    report [LIBRARY."LibraryName" ] ;
```

references the DRIVE and LIBRARY clusters. The derived constraint between libraries and drives is that the drive in the current library-drive combination being examined is resident in the library of the library-drive combination being examined.

For another example, the following command:

```
attribute task ["2"] match [streq (APPLICATION."color" "green")]

    set [VOLUME."color" "blue" ] ;
```

The tables mentioned in the example above are from the CARTRIDGE and APPLICATION cluster. The derived constraint introduced is that while examining cartridge-application combinations, the MM includes only those combinations where the cartridge being examined is in a CARTRIDGEGROUP that is accessible to the application being examined.

4.1.4 Determination of working set

The determination of the eventual working set involves the following ordered steps:

- a) Determine the combination of objects to which the constraints should be applied. The object types examined are the canonical list of all objects in the constraint set (i.e., the tables in the implied constraint set, the tables in the explicit constraint set, and the tables in the derived constraint set).
- b) Simultaneously evaluate all constraints (implied, explicit, and derived) on each combination of objects, eliminating those combinations that do not satisfy the constraint expressions.
- c) The resulting set of objects may be sorted. Some commands implicitly order their results; many commands permit an **order** clause that specifies explicit ordering. If neither implicit nor explicit ordering is performed, the result is unordered. If both implicit and explicit ordering occur, the implicit ordering is the primary sort key and the explicit ordering is the secondary key.
- d) Optionally, select from the ordered set, elements that are specified in **number**. Additional restrictions are applied with regards to **number**, and are enumerated in the command descriptions that follow.

In the absence of **number**, any element of the working set may be chosen. If, however, **number** specifically identifies more than one element of the object in question, and if the command only operates on one object at a time, then the command is in error.

4.1.5 View limitations on non-privileged applications

Table 2 lists “view” limitations that the MM applies on a command when issued by a non-privileged application. Note that the constraints dictate inclusion criteria. Also note that the constraints apply to all containing objects. For example, if **match** in an **attribute** command refers to the **SIDE** object, then the MM allows inclusion (into the working set) of only those **SIDE** objects that are defined on cartridges into which the application has visibility.

Table 2—Command view limitations

Object cluster	Constraint
CARTRIDGE	Limited to accessible cartridges, for example, cartridges that contain volumes that are owned by this application, and un-owned cartridges in cartridge groups that are accessible to this application (CARTRIDGEGROUPAPPLICATION).
DRIVE	Limited to drives in accessible drive groups (DRIVEGROUPAPPLICATION).
LIBRARY	Limited to libraries that contain cartridges and/or drives for which the application has visibility.
APPLICATION	Limited to itself. Information regarding other applications is not visible.
TASK	Limited to tasks issued by this application only, including tasks issued by other instances of this application.
SESSION	Limited to all sessions created by all instances of this application only.
MOUNT	Limited to cartridges mounted by instances of this application only.

4.2 MMP Responses and command sequencing

Table 3 lists the protocol sequence, as described in IEEE Std 1244.1-2000.

Table 3—Protocol sequence

Command	Described under each command in Clause 5.
Acknowledgement	The content and format of the acknowledgement is described in IEEE Std 1244.1-2000. It indicates that the receiving program has or has not successfully received and parsed the command.
Response	The format of the response is described in IEEE Std 1244.1-2000. As noted in that standard, if there are errors in the execution of the command, these errors are returned with the response. In addition, some commands, when executed successfully, contain additional parameters in the response. The specific error codes for this protocol for each command are described with the command in Clause 5.

The result of sending an MMP command is two communicated sequences. The first is the acknowledgement that shall be received by the client before it sends another command, the second is the response that may be

sent by the MM anytime after the command is complete and after MM has sent the acknowledgment. Both of these sequences contain the text “response.” The EBNF is as follows:

```

<response-message> ::= "response" { <negative-response> |
                                   <accepted-response> |
                                   <command-response> } ";"

<negative-response> ::= { "unacceptable" [ <message-clause> ] }

<accepted-response> ::= { <task-clause> "accepted" }

<command-response> ::= { <task-clause>
                        <final-response> [ <message-clause> ] }

<final-response> ::= <response-success> |
                    <response-cancelled> |
                    <response-error>

<response-success> ::= { "success" <response-text>* }

<response-cancelled> ::= "cancelled"

<response-text> ::= "text" "[" <text-string>+ "]"

<task-clause> ::= "task" "[" <task-identifier> "]"

<response-error> ::= "error" "[" <class> <code> "]"

<message-clause> ::= "message" "[" { <id-clause> [ <arg-clause> ]
                                   <locale-text-clause>* } "]"

<id-clause> ::= "id" "[" <manufacturer> <model> <messageid> "]"

<arg-clause> ::= "arguments" "[" <arg-pair>+ "]"

<arg-pair> ::= <arg-key> <arg-text>

<locale-text-clause> ::= "loctext" "[" <language> <localized-string> "]"

```

4.2.1 Error responses

The general response format described in IEEE Std 1244.1-2000 also applies to responses in the MMP. Some special notes apply to error responses from the MM.

The MM will only generate a single error clause in an error response to an MMP command, even if the command encountered multiple errors. For example, the MM-implementation may choose to report the most appropriate error or the most recent error.

The **error** clause includes information that an application (as opposed to a human) may use to determine the nature and cause of the error and also to dynamically determine a retry or recovery strategy. The error **code** and error **class** tokens in the **error** clause define a two-level hierarchy. This standard describes the complete set of error **classes** that an MM may generate. This version of the standard also defines sets of error **codes**, for each error **class**. An MM implementation may supplement the set of error **codes** through the registry process.

In addition to the standard-defined error **codes**, each error **class** may have additional vendor-defined error **codes**. An application must be prepared to accept both standard-defined error **codes** as well as vendor-defined error **codes**. An MM implementation may not extend the set of error **classes**, though.

This standard does not define the textual messages that are part of the error response. Special rules apply to localization of textual error messages. Refer to the **locale** command (6.2.2) for more information on this.

4.3 Error model

In order to describe the error reporting structure of the MM, the standard defines an execution model on which the error reporting structure is based. Figure 1 illustrates the execution of MMP commands. Note that this description does not imply, nor does it suggest or dictate that an MM be implemented exactly as described, but simply states that the MM implementation shall adhere to the error semantics that result from this model.

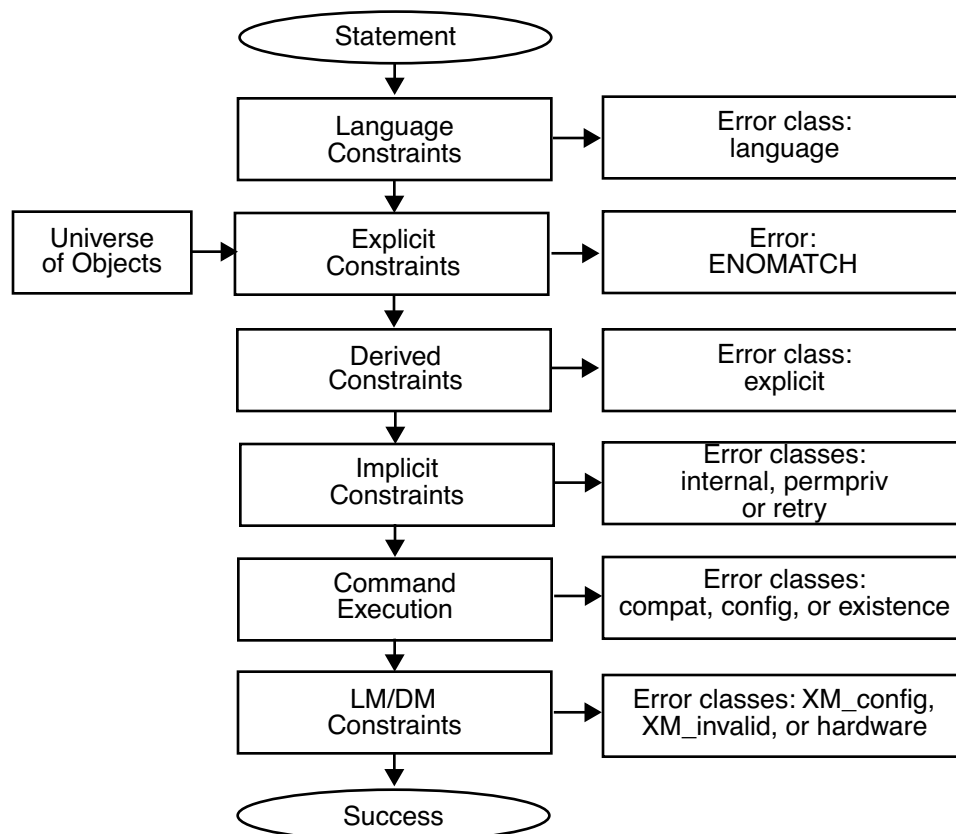


Figure 1—Execution model

Every error response from the MMS includes both an error class and an error token. Each error token falls into exactly one of several standards-defined error classes. Error classes facilitate general recovery strategy for an application, whereas error codes dictate specific recovery. An MM (or LM/DM) implementation may define error codes in addition to the ones that this standard defines.

The execution of an MMP command may be viewed as a sequence of ordered phases, wherein each phase may result in one or more errors. The phases are as follows:

- a) Examination of the command as submitted by the application may result in an error. Errors generated during this phase are in the language error class.
- b) The MM starts with the universe of objects and determines a set of objects to which it shall apply the explicit constraints specified in the command. This combination of objects includes all of the objects explicitly mentioned in the command and the target objects for that command. (Refer to 4.1.1.1 for a listing of the target objects for each command.) If no combinations of objects remain after evaluation of the explicit constraints, the MM implementation shall return ENOMATCH. This indicates that the explicit constraints as specified cannot be satisfied. The MM shall return ENOMATCH only in this circumstance. The error class for the ENOMATCH error code is the explicit error class.

For example, the MMP command:

```
allocate newvolname[ "vol-003" ]
    match[ and( streq( LIBRARY. "color" "green" )
                streq( APPLICATION. "color" "blue" )
                streq( DRIVE. "color" "pink" ) ) ];
```

will examine all combinations of the LIBRARY, CARTRIDGE, and PARTITION objects that exist in the system, while applying the three explicitly specified constraints. If no combinations of objects satisfy the explicit constraints, then the ENOMATCH error token is returned.

- c) If some combination of objects remain, the MM then applies the derived constraints. Recall, as described in 4.1.3, these constraints allow the MM to arrive at the target objects when, seemingly, unrelated objects are specified in the command. Multiple constraints may be involved and the MM may apply the derived constraints in any order. However, the order shall be consistent for an MM implementation from one command to the next. If no combinations remain after the application of any one of the derived constraints, then the MM returns a specific error code, i.e., each derived constraint has a unique error code. The error class for each of the error codes generated during this phase is the explicit error class.

Continuing the previous example, the following derived constraints are applied:

- 1) The application being examined shall have access to the library being examined. Either the library shall contain one or more drives that the application has access to via a DRIVEGROUPAPPLICATION object or the application shall have access to one or more cartridges contained in the library. For non-privileged applications, there shall be a cartridge owned by that application in this library or there shall be an unowned cartridge in this library that the application has access to via a CARTRIDGEGROUPAPPLICATION object.
- 2) The drive being examined shall reside in the library being examined.
- 3) The drive being examined shall be accessible by the application being examined via a DRIVEGROUPAPPLICATION object.

The MM applies each of these constraints one by one on all the objects. If no combinations remain after application of a constraint, then the MM reports an error code.

- d) Each command specifies an ordered list of implicit constraints that shall be satisfied in order for the command to succeed. The MM applies these implicit constraints one by one on all the combinations

of objects that remain in the working set. If no combinations remain after application of an implicit constraint, then the MM reports a specific error code. Unlike the previous phase where no ordering is defined, the standard defines an ordering on the error codes that arises due to the application of the implicit constraints. This ordering defines which errors are returned to the application by the MM when one or more of the implicit constraints are not satisfied. An MM implementation need not necessarily apply the constraints in any particular order; however, error reporting shall adhere to this ordering. The error codes returned during this phase of execution of the MMP command may fall into any of the other error classes (i.e., other than the language and explicit error class). The Error Responses clause (i.e., Error conditions) in the description for each MMP command enumerates the standard defined error tokens in the required reporting order.

Continuing the example, the allocate command brings in the following implicit constraints (assume the application issuing the command is non-privileged):

- The cartridge being examined shall be accessible to the application making the request (potentially different from the application in the combination being examined) via a CARTRIDGEGROUPAPPLICATION object.
 - The application making the request shall own the cartridge or the cartridge shall not be owned by any application.
 - There shall be an allocable partition on one of the cartridges being examined.
- e) After identifying the objects to operate upon, the MM executes the command. This may include performing one or more suboperations, each of which may result in errors in different error classes. If no errors occur, the command returns successfully.

Table 4 describes the error classes.

Table 4—Error classes

Error class	Component where this error may be generated	Description and recovery
Language	MM	Indicates that the command as submitted is syntactically incorrect. The command has been rejected without any processing. For example, a missing clause that is required for this command may be responded to with an error that falls in this class.
Explicit	MM	Application of the explicit constraints and derived constraints imply that the command as submitted can never be satisfied by the system at the time of evaluation of the command. There is no expectation that vendors will define extended error tokens in this error class.
Invalid		
LM_invalid		
DM_invalid	MM	
LM		

Table 4—Error classes (continued)

Error class	Component where this error may be generated	Description and recovery
DM		Errors in this class indicate that although the command as submitted is syntactically correct, some argument or value to a clause was invalid. For example, an attribute command that attempts to modify a numeric system-defined attribute with a non-numeric value.
Config		
LM_config		
DM_config	MM	
LM		
DM		Indicates that the current static configuration prevents the operation from succeeding. For example, a missing configuration file for an LM.
Retry	MM	Indicates that current transient state of the system prevents the operation from succeeding, and retrying at a later time is suggested. For example, attempts to mount without blocking a cartridge that is currently in use by another application are flagged with this error class.
Internal		
LM_internal		
DM_internal	MM	
LM		
DM		Internal or system error, or unexpected or inconsistent state. Specific to the implementation.
Permpriv		
LM_permpriv		
DM_permpriv	MM	
LM		
DM		Indicates a violation of a system or site imposed restriction.
Compat	MM	Indicates a mismatch between what is requested and what is currently offered.
Existence	MM	Existence or lack of existence (usually of an entity in the data model) prevents the operation from succeeding. For example, an attempt to create a LIBRARY instance that already exists.
Subop	MM	Some suboperation failed. Generally specific to the implementation.
LM_hardware		

Table 4—Error classes (continued)

Error class	Component where this error may be generated	Description and recovery
DM_hardware	LM	
DM		Indicates a failure related to the underlying device.
LM_media		
DM_media	LM	
DM		Errors related to physical media.

The description of each MMP command in Clause 5 includes a subclause (Command error responses) listing the typical errors that may generated during the execution of that command. The errors listed in these subclauses are by no means exhaustive. Errors falling in the language and explicit error classes are not shown. The errors shown there are typically generated during the application of implicit constraints and errors that arise from execution of the suboperations that this command entails. However, the error tokens shown in these subclauses are in the order in which an MM reports them. An MM implementation may augment the list of errors that may arise during the execution of this command, and may in fact add new error tokens, but they shall follow the standard defined errors in terms of priority of reporting.

5. Semantics of common syntactic elements

There are several syntactic elements that are common to many MMP commands. These include **match**, **order**, **number**, **report**, **reportmode**, and others. All commands are designed to allow the basic syntactic elements to be specified in any order.

The **order** clause serves a dual purpose. In addition to serving the function of sorting elements in the working set, the tables mentioned in **order** influence the derived constraints that are used during the evaluation of the command.

NOTE—The following clauses include subclauses listing the syntax of each common syntactic element in Extended Backus-Naur Form (EBNF). Some commonly used EBNF non-terminal symbols have been excluded for the sake of brevity. Refer to the *Language Descriptions: Extended BNF* clause of IEEE Std 1244.1-2000 (Clause 1.2 or IEEE Std 1244.1-2000) for the production rules of the remaining non-terminal symbols.

5.1 Syntactic element name: OBJECT."attributename"

The OBJECT."attributename" syntax is how a particular attribute of an object is referenced. This syntax is used mostly in the **match**, **order**, and **report** syntactic elements. It is composed of an object type name chosen exclusively from the list of predefined tables, and an attribute name. The attribute name may be a system-defined attribute name, or it may be a user-defined attribute name.

This construct is used in the following three places in the language:

- Viewing (i.e., in a **report** clause)
- As an argument to a comparison or ordering operator (i.e., in the **match** or **order** clauses)
- Assignment of value to an attribute (i.e., in a **set** or **unset** clause)

The following examples show the use of the OBJECTNAME."attributename" construct in **report** and **match** clauses.

Example:

```
show task["1"] volname["servers.001"] report[CARTRIDGE."CartridgePCL"];

show task["1"] match[streq(DRIVE."DriveName" "physics1") ]

    report[LIBRARY."LibraryName" ] ;
```

The first command example reports the PCL(s) of the cartridges of all the volume(s) named "servers.001" from all applications, while the second command example reports the name of the library in which the drive "physics1" resides.

5.1.1 EBNF for OBJECT."attributename" clause

<object-attribute> ::= <object-name> "." <attribute-name>

<object-name> ::= APPLICATION | AI | BAY | CARTRIDGE |
 CARTRIDGEGROUP | CARTRIDGEGROUPAPPLICATION |
 CARTRIDGETYPE | CONNECTION | DM | DMBITFORMAT |
 DMBITFORMATTOKEN | DMCAPABILITY | DMCAPABILITYTOKEN |
 DMCAPABILITYDEFAULTTOKEN | DMCAPABILITYGROUP |
 DMCAPABILITYGROUPTOKEN | DRIVE | DRIVEGROUP |
 DRIVEGROUPAPPLICATION | DRIVECARTRIDGEACCESS | LM |
 LIBRARY | MESSAGE | MOUNTLOGICAL | MOUNTPHYSICAL |
 PARTITION | REQUEST | SESSION | SIDE | SLOT |
 SLOTCONFIG | SLOTGROUP | SLOTTYPE | STALEHANDLE |
 SYSTEM | TASK | TASKCARTRIDGE | TASKDRIVE |
 TASKLIBRARY | VOLUME

<attribute-name> ::= <string>

5.2 Clause name: volname

volname is a convenient shorthand means of expressing **match** involving string equality of one or more VOLUME."VolumeName" attributes. This clause accepts one or more strings, each of which is treated as the value for the VOLUME."VolumeName" attribute. **volname** is mutually exclusive with **match** and at most one **volname** may be specified per command.

Example:

The following construct:

```
volname[ "servers.001" "servers.002" "servers.003" ]
```

is equivalent to

```
match[ or( streq( VOLUME."VolumeName" "servers.001")
          streq( VOLUME."VolumeName" "servers.002")
          streq( VOLUME."VolumeName" "servers.003")) ]
```

5.2.1 EBNF for volname clause

```
<volname-clause> ::= "volname" "[" <volume-name>+ "]"
```

```
<volume-name> ::= <string>
```

5.3 Clause name: match

An application may explicitly enumerate constraints on an MMP command. These constraints shall hold true for any solution that the MM chooses for that command. It is important to remember that explicit constraints are not the sole determinant of whether a combination of object instances is included in the working-set of the command or not; but explicit constraints are a necessary condition. It is also important to note that since **match** specifies explicit constraints, the objects that are mentioned in the clause are all included in the combination of objects being examined.

match takes an arbitrary Boolean expression as its argument. This expression is evaluated for each combination of objects being examined. The expression must evaluate to the string representation of the Boolean value "true", for a particular combination to be included in the working-set. Expressions are constructed using various pre-defined subfunctions, object-attributes, and literal constants. Expressions may be of any arbitrary length, and by nature, may be recursive.

Example:

If the current set of volumes being examined and their attributes is as listed in Table 5,

Table 5—Hypothetical volume name and attributes for match

Volume name	Volume Attr (attr=value)	Volume Attr (attr=value)
"vol1"	Group = "Servers"	Handler = "Marge"
"vol2"	Group = "Clients"	Handler = "Sam"
"vol3"	Group = "Servers"	Handler = "Bill"
"vol4"	Group = "Clients"	Handler = "Marge"

then the following fragment:

```
match[ and (streq (VOLUME."Group" "Servers")
              strne (VOLUME."Handler" "Marge"))]
```

```
report [ VOLUME."VolumeName" ]
```

will yield the result

```
vol3
```

If presented in English, the **match** presented above would read: Find the volumes where the "Group" attribute is set to "Servers" and the "Handler" attribute is not "Marge." In this example, after **match** is evaluated, only those combinations of objects, which include the volume named "vol3," remain in the working set.

5.3.1 EBNF for match clause

```
<match-clause> ::= "match" "[" <basematch> "]"

<basematch> ::= <unaryattrop>
               | <unarynegop>
               | <topmatch>

<unaryattrop> ::= ("isattr" | "noattr") "(" <object-attribute> ")"

<unarynegop> ::= "not" "(" <matchspec> ")"

<topmatch> ::= <binaryop> "(" matchspec matchspec ")"
              | <multiop> "(" matchspec+ ")"

<matchspec> ::= <topmatch>
               | <unaryattrop>
               | <unarynegop>
               | <objectRef>
               | <string>
               | <number>

<binaryop> ::= "regex"
              | "streq"
              | "strne"
              | "strlt"
```

```

| "strle"
|
| "strgt"
|
| "strge"
|
| "numeq"
|
| "numne"
|
| "numlt"
|
| "numle"
|
| "numgt"
|
| "numge"

<multiop> ::= "and" | "or"

```

5.4 Clause name: order

The **order** clause is used to sort the working set of objects that remain after the evaluation of all constraints, implied, explicit, or derived. The **order** clause is useful when multiple possible solutions exist for a given command and an application may want to prioritize the eligible candidates in the solution space on some criteria. Multiple sort criteria may be specified and each criterion is honored in a left-to-right fashion, i.e., the first sort criteria is the leftmost sort subfunction specified in the **order** clause, and so on. Only one **order** clause may be specified per command. For the **strlohi** and **strhilo** operations, the sort order that is currently in force, as dictated by the current locale, is employed while performing the ordering.

Sort criteria are specified using any of the following ordering functions: **strlohi**, **strhilo**, **numlohi**, **numhilo**. The argument to each of these functions is an object-oriented combination.

- a) **strlohi**: The resulting string is sorted in an increasing order, defined by the current locale.
- b) **strhilo**: The resulting string is sorted in a decreasing order, defined by the current locale.
- c) **numlohi**: The arguments to this sort function are treated as numbered and sorted in numerically increasing order.
- d) **numhilo**: The argument to this sort function are treated as numbered and sorted in numerically decreasing order.

The native data type supported by the system is a string. Values that represent non-string quantities are converted to the appropriate data type depending on the command being performed and the locale in which the command is being executed.

The objects mentioned in **order** are all included in the combination of objects being examined. The values for user-defined attributes that represent numeric quantities shall be expressed in a canonical fashion in order for the system to treat the value of the attribute as a numeric value. This requirement also holds for numeric constants input by the application.

Refer to the Data Model for the representation of system-defined attributes that reflect numeric quantities, or of attributes that are enumerated data types, or of attributes that are time-stamps. Refer to 5.10 for details on how non-existent/invalid operands are treated.

Example:

If the current working set of objects is defined by the set of volumes and their attributes that are listed in Table 6,

Table 6—Hypothetical volume name and attributes for order

Volume name	Volume Attributes (attr=value)
"vol1"	PctFull = "40"
"vol2"	PctFull = "31"
"vol3"	PctFull = "93"
"vol4"	PctFull = "11"

and the **order** clause is as follows:

```
order[ numhilo (VOLUME."pctFull") ]
```

then the order of volumes in the working set will be

```
vol3 vol1 vol2 vol4
```

5.4.1 EBNF for order clause

```
<order-clause> ::= "order" "[" <order-one> "]"
```

```
<order-one> ::= <order-op> "[" object-attribute "]"
```

```
<order-op> ::= "strlohi"
```

```
      | "strhilo"
```

```
      | "numlohi"
```

```
      | "numhilo"
```

5.5 Clause name: number

number is used to declare which of the elements in the current set of objects being examined are to be used as candidates for the current command. Only one **number** may be specified per command.

The elements inside **number** specify the ordinal numbers of the items in the (possibly sorted) work list being operated on. It is possible to specify both single items and ranges of items. A range is given by two numbers separated by a pair of periods (".."), and is inclusive of the elements at each end of the range. There

are two additional tokens, "FIRST" and "LAST", that refer to the first and last elements in the work list, respectively. Negative numbers are ordinal offsets from the end of the sorted list of items in the work list.

If the work list has not been sorted as a result of an implicit or explicit ordering then **number** will return items from an unordered list.

NOTE—Numbers equal to zero are not allowed, nor can the low value be greater than high value.

Examples:

```
number[1 3 5]
```

means that the first, third, and fifth items from the ordered work list should be used.

```
number[2..4]
```

is exactly equivalent to

```
number[2 3 4]
```

```
number[FIRST..3 7..-8 -3..LAST]
```

when there are 16 items in the work list, is exactly equivalent to

```
number[1 2 3 7 8 9 14 15 16 ]
```

Example:

If the current working set of objects is defined by the set of volumes and their attributes that are listed in Table 7,

Table 7—Hypothetical volume name and attributes for number

Volume name	Volume Attr (attr=value)	Volume Attr (attr=value)
"vol1"	Group = "Servers"	Handler = "Marge"
"vol2"	Group = "Clients"	Handler = "Sam"
"vol3"	Group = "Servers"	Handler = "Bill"
"vol4"	Group = "Clients"	Handler = "Marge"

and the **number**, **report**, and **reportmode** clauses are as follows:

```
number[2 4]
```

```
report[VOLUME."group" VOLUME."VolumeName" VOLUME."handler"]
```

```
reportMode[value]
```

then the output of the command will be

```
text["Clients" "vol2" "Sam"]
```

```
text["Clients" "vol4" "Marge"]
```

5.5.1 EBNF for number clause

```

<number-clause> ::= "number" "[" <number-spec>+ "]"

<numberspec> ::= <number-range> | <number>

<number-range> ::= <number> ".." <number>

<number> ::= <nonzero-ascii-number> | "FIRST" | "LAST"

<nonzero-ascii-number> ::=

    [<sign>] <ascii-digit>* <nonzero-ascii-digit> <ascii-digit>*

<sign> ::= "+" | "-"

<ascii-digit> ::= "0" | <nonzero-ascii-digit>+

<nonzero-ascii-digit> ::= "1"

    | "2"

    | "3"

    | "4"

    | "5"

    | "6"

    | "7"

    | "8"

    | "9"

```

5.6 Clause names: report, reportmode, cpreport, and cpreportmode

report is used to declare the attributes and/or attribute names values that are to be returned by the current command. **reportmode** declares whether the output from **report** contains only the name of each attribute to be reported ("*name*"), only the value of the attribute ("*value*"), or both the name and value ("*unenviable*"), and controls whether all combinations are returned or only the unique combinations ("*unique*"). When the **reportmode** clause is specified, the order of these parameters doesn't matter. The **report** and **reportmode** clauses together describe the required format of the command output. When a reportmode clause is not specified, the default is "value."

The order of values returned in the response to a command is the same order in which the attributes are listed in **report**. **report** also allows just an object name to be specified, i.e., with no attribute names. In this case, the MM will apply **reportmode** to every attribute defined on the object, including user-defined attributes and the reported names/values are delimited using the **attrlist** subclause of the **text** clause.

cpreport and **cpreportmode** behave exactly like the **report** and **reportmode** except they cause the **private** command to be sent to an LM or DM, and refer to objects not in the data model.

Example:

```
show task ["2"] report[DRIVE LIBRARY] reportmode[namevalue];

response task["2"] accepted;

response task["2"]

    text[

        attrlist["DriveName" "drive1" "DriveGroupName" "drives ... ]

        attrlist["LibraryName" "library-1" "LMName" "lib1_lcp" ...]]

    text[

        attrlist["DriveName" "drive2" ... ]

        attrlist["LibraryName" "library-2" ... ]];
```

Only one **report** and one **reportmode** may be specified. A **reportmode** with no **report** is ignored. If no **report** is specified, the response to a command that was successfully executed will not include anything outside of a normal “success” response.

Example:

If the current working set of objects is defined by the set of volumes and their attributes that are listed in Table 8,

Table 8—Hypothetical volume name and attributes for report and reportmode

Volume name	Volume Attr (attr=value)	Volume Attr (attr=value)
"vol1"	Group = "Servers"	Handler = "Marge"
"vol2"	Group = "Clients"	Handler = "Sam"
"vol3"	Group = "Servers"	Handler = "Bill"
"vol4"	Group = "Clients"	Handler = "Marge"

and **report** and **reportmode** are as follows:

```
report[VOLUME."group" VOLUME."VolumeName" VOLUME."handler"]

reportmode[value]
```

then the output is

```
text["Servers" "vol1" "Marge"]

text["Clients" "vol2" "Sam"]
```

```
text["Servers" "vol3" "Bill"]
```

```
text["Clients" "vol4" "Marge"]
```

If **reportmode** is changed to

```
reportmode[nameValue unique]
```

then the output is

```
text["group" "Servers" "VolumeName" "vol1" "handler" "Marge"]
```

```
text["group" "Clients" "VolumeName" "vol2" "handler" "Sam"]
```

```
text["group" "Servers" "VolumeName" "vol3" "handler" "Bill"]
```

```
text["group" "Clients" "VolumeName" "vol4" "handler" "Marge"]
```

If **reportmode** is changed to

```
reportMode[unique name]
```

then the output is

```
text["group" "VolumeName" "handler"]
```

5.6.1 EBNF for report and reportmode clauses

<report-clause> ::=

```
"report" "[" (<object-attribute>|<object-name>)+ "]"
```

<reportmode-clause> ::= "name"

```
| "namevalue"  
| "value"  
| "name" "unique"  
| "unique" "name"  
| "namevalue" "unique"  
| "unique" "namevalue"  
| "value" "unique"  
| "unique" "value"
```

5.7 Clause name: text

text is a general container for lists of character strings or object references. In some cases the number of strings that can be enclosed by text may be constrained. In most cases, though, one or more **text** clauses are used to encapsulate the response from a command.

Example:

```
response success text["blue" "red" "green" "yellow"];
```

Please refer to 5.6.1 defining **report** and **reportmode** for examples of the use of **text** as a command output construct.

5.7.1 EBNF for text clause

```
<text-clause> ::= "text" "[" <string-list> "]"
```

```
<string-list> ::= (<string>|<sub-string>)+
```

```
<sub-string> ::= "attrlist" "[" <string>+ "]"
```

5.8 Clause name: reqid

reqid is similar to volname. It is a convenient shorthand means of expressing a **match** that involves string equality of one or more REQUEST."RequestID" attributes. **reqid** accepts one or more strings, each of which is treated as the value for the REQUEST."RequestID" attribute. **reqid** is mutually exclusive with **match** and at most one **reqid** may be specified per command.

Example:

If there are four requests currently in the MM, such as appears in Table 9,

Table 9—Hypothetical request IDs and states for reqid

Request ID	Request state
"53"	Pending
"54"	Pending
"55"	Accepted
"255"	Pending

then the following **accept**:

```
accept task["dmf009"] reqid["53" "54" "55"] number[1..3];
```

is equivalent to

```
accept task["dmf009"] match[ or (
    streq(REQUEST."RequestID" "53")
```