

IEEE Standard for Media Management System (MMS) Architecture

Sponsor

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

Approved 21 June 2000

IEEE-SA Standards Board

Abstract: The architecture of a distributed, platform-independent system that manages removable media, including both disk and tape, using robotic and manual methods are specified. The general schema for managing media, the components of the software system, and the supporting data model used by the software system for managing this media are described by this standard. Details of major components of the system are specified by companion standards.

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 23 October 2000. Printed in the United States of America.

Print: ISBN 0-7381-2506-7 SH94862
PDF: ISBN 0-7381-2507-5 SS94862

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 1244.1-2000, IEEE Standard for Media Management System (MMS) Architecture.]

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 P1244.2/D041900—Draft 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 Architecture standard (IEEE Std 1244.1-2000) is best understood in the context of the others by reading the first sections of the Architecture. 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.^a 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

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

IEEE P1244.2/D041900—Draft 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

George B. Cole

John L. Cole

Don Doerner

Richard T. Eberlein

Robert J. Gagliano

Mark R. Gary

Michael V. Hardy

P. C. Hariharan

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.

HACMP is a trademark of the International Business Machines 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	1
1.2	Purpose	3
1.3	Conformance	4
2.	References	5
3.	Definitions, abbreviations, and acronyms	5
3.1	Definitions	5
3.2	Acronyms and abbreviations	7
4.	Functionality	7
4.1	Interface protocols	8
4.2	Programming and command line interfaces	9
4.3	Data transfer protocol and interface	10
4.4	Operation of the MMS	10
4.5	LM operational overview	16
4.6	System start-up, restart, and shutdown	18
5.	Operation and matching model	18
6.	System model	19
6.1	Media	19
6.2	Applications	23
6.3	Media Manager (MM)	28
6.4	Library Managers (LMs) and Drive Managers (DMs)	29
7.	Protocols	30
7.1	Language style	30
7.2	Language descriptions: extended BNF	31
7.3	Message sequencing within protocols	32
7.4	Security and authentication	33
7.5	Internationalization	33
7.6	Overview of SSAIP	37
7.7	Overview of MMP	37
7.8	Overview of DMP	38
7.9	Overview of LMP	38
7.10	Overview of MMIP	39
7.11	Overview of MMCIP	39
8.	Tokens	39
9.	Data model—introduction	39

9.1	Clusters and objects	40
9.2	Relationships between clusters	41
9.3	Data object list	44
10.	Privilege levels, attribute types, and data types	46
10.1	Application privilege levels	46
10.2	Attribute types	46
10.3	Summary of privileges and attribute types	48
10.4	Data types	48
10.5	Default values	49
11.	Object types	49
12.	Application cluster	50
12.1	Creation and deletion semantics	50
12.2	APPLICATION object	50
12.3	AI object	52
13.	The library cluster	54
13.1	Creation and deletion semantics	54
13.2	The LIBRARY object	55
13.3	The LM object	57
13.4	BAY object	60
13.5	The SLOT object	61
13.6	SLOTGROUP object	64
13.7	SLOTCONFIG object	66
13.8	SLOTTYPE object	68
14.	Drive cluster	69
14.1	Creation and deletion semantics	70
14.2	DRIVE Object	71
14.3	DRIVEGROUP object	77
14.4	DRIVEGROUPAPPLICATION object	78
14.5	DM object	79
14.6	DMBITFORMAT object	82
14.7	DMBITFORMATTOKEN	83
14.8	The DMCAPABILITYobject	85
14.9	The DMCAPABILITYTOKEN object	86
14.10	DMCAPABILITYDEFAULTTOKEN	87
14.11	DMCAPABILITYGROUP object	88
14.12	DMCAPABILITYGROUPTOKEN	90
15.	Cartridge cluster	91
15.1	Creation and deletion semantics	91
15.2	The CARTRIDGE object	93
15.3	The CARTRIDGEGROUP object	97
15.4	The CARTRIDGEGROUPAPPLICATION object	98
15.5	The CARTRIDGETYPE object	99
15.6	The SIDE object	102

15.7	The PARTITION object	104
15.8	The VOLUME object	110
16.	The mount cluster	113
16.1	The MOUNTLOGICAL object	113
16.2	The MOUNTPHYSICAL object	116
16.3	The DRIVECARTRIDGEACCESS object	119
17.	The session cluster	124
17.1	The CONNECTION object.....	124
17.2	The SESSION object	126
18.	The task cluster	129
18.1	The TASK object	130
18.2	The TASKCARTRIDGE object	132
18.3	The TASKDRIVE object.....	133
18.4	The TASKLIBRARY object.....	134
19.	The system cluster.....	135
19.1	The MESSAGE Object	135
19.2	The REQUEST object.....	137
19.3	The SYSTEM object.....	141
19.4	The STALEHANDLE object.....	146
20.	Tokens	148
20.1	Instructions	148
20.2	MMP mount modes.....	149
20.3	Slot type names	150
20.4	Cartridge shape names	151
20.5	Bit formats	152
20.6	Cartridge type names	154
20.7	Partition names.....	155
20.8	Attribute names	156

IEEE Standard for Media Management System (MMS) Architecture

1. Overview

The IEEE Media Management System (MMS) is a distributed, multi-platform system for managing removable media. The IEEE MMS standards define a software component model for working with removable media as well as a number of protocols that define interfaces between the components. These standards enable vendors to construct applications that use removable media as well as components of an MMS that interoperate with other MMS components.

Systems based on the MMS may support many kinds of removable media, including computer media such as magnetic tape, optical disk, and CD-ROM, as well as non-computer media such as audiotape and videotape, film, and audio CDs and videodiscs. The MMS supports both automated (robotic) and non-automated handling of media.

The MMS is scalable and both system and media neutral. Systems that implement the MMS may target a range of customers, from individual desktop computers, to workgroups, to data centers, to multinational corporations. The MMS does not require any particular hardware or operating system, and allows organizations that have diverse systems to manage the removable media that is used by these systems in a unified manner. The media managed by the MMS does not need to be formatted or labeled for the MMS; existing media can be incorporated into an MMS-managed system with no modifications. The MMS utilizes existing machine-readable label information such as bar codes or media labels if they are present.

The MMS and widely available implementations of the MMS will give users of removable media greater security and convenience of operation. Software developers will be able to write software that takes advantage of the huge storage capacity of removable media much more easily, and this software will be portable to a wide number of platforms. System and component vendors will be able to incorporate the MMS into their systems, allowing greater interoperability and eliminating the need to implement costly, proprietary solutions to manage removable media.

1.1 Scope

The MMS standards published by the IEEE define the architecture, data model, and interfaces that are required in any MMS implementation. These standards are not intended as a tutorial or textbook, nor do they provide a cookbook for implementation of an MMS. The standards are organized to be of use to several audiences:

- a) Those interested in an introduction to the MMS and its capabilities will find this standard, IEEE Std 1244.1-2000, to be a reasonable, though technical, introduction to the system.
- b) Those interested in implementing applications that utilize the MMS will find this standard, IEEE Std 1244.1-2000, and IEEE Std 1244.3-2000¹ to be helpful.
- c) Those interested in implementing devices such as robotic libraries or drives will find IEEE Std 1244.1-2000, IEEE P1244.2/D041900, IEEE Std 1244.4-2000, and IEEE Std 1244.5-2000, to be essential.
- d) Those interested in implementing the entire MMS will find all standards in the IEEE MMS suite of standards to be required reading.

A suite of IEEE 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 1244.1-2000, 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 MMS, its associated standards, and the components of an MMS implementation.

The set of standards that comprise the IEEE MMS suite are listed in Table 1.

Table 1—The IEEE MMS suite of standards

Standard	Description	Acronym	Document
System Architecture			
IEEE Std 1244.1-2000	IEEE Standard for Media Management System (MMS) Architecture	MMS	available
Native Protocols			
IEEE P1244.2/D041900	Draft Standard for Session Security, Authentication, Initialization Protocol	SSAIP	available
IEEE Std 1244.3-2000	IEEE Standard for Media Management Protocol	MMP	available
IEEE Std 1244.4-2000	IEEE Standard for Drive Management Protocol	DMP	available
IEEE Std 1244.5-2000	IEEE Standard for Library Management Protocol	LMP	available
Interoperability Protocols			
P1244.6	Draft Standard for Media Manager Interchange Protocol	MMIP	—
P1244.7	Draft Standard for Media Manager Control Interface Protocol	MMCIP	—
Programming and Command Line Interfaces			
P1244.8	Draft Standard for C Language Procedural Interface	—	—
P1244.9	Draft Standard for User Mount Commands	—	—
P1244.10	Draft Standard for Standard Administrative and Operational Commands	—	—
Data Transfer Protocol			
P1244.11	Draft Standard for Media Data Mover	—	—

¹Information on references can be found in Clause 3.

Readers should be aware that these standards are primarily reference documents, and as such, may be organized in a manner that is not conducive to sequential reading. (Feel free, for example, to skim or skip ahead of the *Definitions* clause, Clause 3 in this standard, as you read.)

The authors of the IEEE 1244 standards expect that additional publications that document aspects of the MMS will appear over time. It is intended that the IEEE MMS suite of standards be considered the definitive reference to the standard, and that these additional publications exist to explain, clarify, and amplify the IEEE standards, aspects of implementation, or of management of MMS sites.

It is expected that vendors will implement proprietary products that add value to systems that are based on the IEEE MMS. These proprietary products must not change the required aspects of the standard as addressed in the IEEE standards. These products may significantly improve the functionality and usability of a standards-based system. Over time, such proprietary extensions may be proposed as additions to the IEEE standard. Until such time as an extension is transformed into an approved IEEE standard, it must not be referred to as a standard.

1.2 Purpose

This standard describes the motivations for an overall architecture of the MMS. Although the architecture may suggest a particular design or implementation, it is not the IEEE's intent to favor a specific implementation of the MMS. Indeed, it should be possible to implement the MMS in a number of ways, ranging from a *lightweight* implementation in a scripting language such as Perl, or a full implementation written in a traditional programming language such as C, C++, or Java.

The MMS is a software system for managing physical media. The system has the following properties:

- a) It is *media neutral* to allow the management of computer tapes, disk media, disks, optical disks, CD-ROMs, as well as non-computer media such as videotapes or reels of film.
- b) It is *scalable* to be comfortable in environments as small as a single individual's office or home and as large as a multinational corporation, educational or scientific institution, or government archives.
- c) It is *platform neutral and operating system independent* to work with existing computer systems from multiple vendors with varying degrees of media-handling sophistication.
- d) It is *distributed* to allow access to media and the devices that store and perform data transfer operations on the media by more than one system. A single MMS may manage devices that are connected to many host computer systems, including devices that are physically connected to multiple hosts. Connectivity between elements of the MMS requires the availability of standard TCP/IP.
- e) It provides a reasonable degree of *security and protection* for access to the media by ensuring that specific media may be mounted only by those applications that have authority to access that media. All parties are authenticated, and network communication is digitally signed so that it is extremely difficult to forge.
- f) It is *content neutral*, and does not have any inherent understanding of the content of the media; indeed, with some media, such as videotape or film, the MMS may not even have access to the content of the media.
- g) It is *application independent* to provide appropriate media management functions for diverse applications ranging from backup and hierarchical storage management, to broadcast television automation. Media belonging to multiple applications may be managed by a single MMS; these applications may be multiple instances of the same program, or of different applications.

- h) It is designed to be *modular* to allow independent groups to work on components of the MMS independently; the modularity is provided by strong, flexible interfaces that can evolve over time.
- i) It is *language neutral* to permit programmers to write applications that interact with the MMS in almost any programming language, and, indeed, to allow the MMS itself to be written in almost any programming language.
- j) It allows *multiple implementations* to interoperate seamlessly.

The key to the architecture of MMS is to clearly define the basic functionality that the MMS must provide, and to declare specific points in the functionality to provide defined interfaces that allow independent components to interoperate.

1.3 Conformance

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.

- a) When a component accepts a defined language or protocol, it shall accept that language or protocol in its entirety—subsetting is not permitted. Note that the languages and protocols are *versioned*, so compliance is with a specific version of a specific language or protocol. In general, a component is expected to start by supporting the current version of the language(s) it supports, and then as new versions of the language are introduced, the component will continue to support older versions.
- b) When a component generates a defined language or protocol, it shall generate language or protocol that fully conforms to the specifications given in this standard—additions, modifications, and extensions are not permitted.
- c) All components must comply with the relevant clauses of the data model as defined in the standard. User-defined attributes and device-specific commands (*cpattribute* and *cpshow*) are permitted within the data model and the languages. These specifically allow a pair of components to incorporate exchanges that are not defined in the languages and protocols, and these are the only mechanisms that may be used to extend the standard environment. This extension mechanism should not be used when an existing mechanism that may be used for the purpose (or a similar purpose) is present in this standard.
- d) If an experimental, non-conforming version of a language is required, this may be done *only* if the language name is changed. It is suggested that such an experimental version be a superset of an existing, defined version of a language. This methodology allows unilateral extension of a language, but the extended language is no longer a part of this standard. This mechanism should be used for experimental implementations of features, which would at some future date be incorporated into a standard version of the language.
- e) To conform to the MMS as a whole, a system shall incorporate the basic software components that are described in this standard [Media Manager (MM), Library Manager (LM), and Drive Manager (DM)] and these software components must have the functions described in these standards. 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 partial conformance with the MMS standard. [For example, a reduced-function MM that fully supports Media Management Protocol (MMP) may incorporate internal implementations of the LM and DM. This is considered partial conformance, because the MM cannot accommodate additional LMs or DMs since it does not implement Library Management Protocol (LMP) or Drive Management Protocol (DMP).]

2. References

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.

ANSI X3.301:1997, Information Technology—SCSI-3 Primary Commands (SPC).²

IEEE P1244.2/D041900, Draft Standard for Media Management System (MMS) Session Security, Authentication, Initialization Protocol (SSAIP).³

IEEE Std 1244.3-2000, IEEE Standard for Media Management System (MMS) Media Management Protocol (MMP).⁴

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 639:1988, Code for the Representation of Names of Languages.⁵

ISO 3166:1993, Codes for the Representation of Names of Countries.

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

ISO/IEC 11578:1996, Information Technology—Open Systems Interconnection—Application Context for Systems Management with Transaction Processing.

IETF RFC 791-1981, Internet Protocol (IP).⁶

IETF RFC 793-1981, Transmission Control Protocol (TCP).

IETF RFC 1766-1995, Tags for the Identification of Languages.

IETF RFC 2279-1988, UTF-8, A transformation format of ISO 10646.

3. Definitions, abbreviations, and acronyms

3.1 Definitions

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

²ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

³This IEEE standards project was not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE.

⁴IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

⁵ISO 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/>).

⁶Internet Requests for Comments (RFCs) are available on the World Wide Web at the following site: www.ietf.org.

3.1.1 administrative application: A program that is concerned with managing operational aspects of the Media Management System (MMS), and typically does not itself *own* media. An administrative application may include an interface for administrative users. Examples of administrative applications include those that allow the addition and removal of applications, drives, libraries, media, and computer systems from the MMS, as well as those concerned with allocation and policy management of an installation.

NOTE—See 6.2.2 in this standard for more information.

3.1.2 automated library: A robotic (electromechanical) library.

3.1.3 cartridge: A unit of physical media that contains one or more sides.

3.1.4 client application: An application program that makes use of Media Management System (MMS) services to manage its media. Examples of client applications include a backup program, a hierarchical storage manager, and an application that allows individual users to mount their own tapes.

3.1.5 compound cartridge: An ordered set of cartridges that may be treated atomically.

3.1.6 data mover (n.): A system program through which a client application accesses the data on media. A data mover is not part of the Media Management System (MMS), nor is a data mover required for operation of the MMS. If a data mover is present, the MMS provides appropriate interfaces and facilities for it to operate.

3.1.7 drive (n.): A piece of hardware that allows access to data that is stored on media. A set of devices such as RAID may be considered as a single drive.

3.1.8 external label: A marking on the exterior of a cartridge that identifies it. The external label may be human-readable, machine-readable, or both. A bar-code label is an example of an external label. *Syn:* **physical cartridge layer.**

3.1.9 internal label: A record in a known format in a known position within a volume that identifies the volume.

3.1.10 library: An automated or manual cartridge-storing facility, e.g., an automated library or a vault. A library may contain zero or more drives, input/output ports and attachments to other libraries.

3.1.11 load (v.): The electromechanical actions of making cartridges ready for access in a drive.

3.1.12 media: Any readable or writable data storage area.

3.1.13 mount (v.): The action of making a cartridge, side, partition, or volume accessible.

- a) Mounting a *volume* is a logical action that implies mounting the one or more partitions that make up the volume.
- b) Mounting a *partition* is a logical action that implies mounting the underlying side of a cartridge and engaging a software interface to gain access to that partition.
- c) Mounting a *side* is the physical action of mounting a cartridge in a drive in a particular orientation.
- d) Mounting a *cartridge* is the physical action of moving a cartridge to a drive and loading it into the drive.

3.1.14 move (v.): The physical action of moving a cartridge from one location to another, where a location is a slot, a drive, or a port.

3.1.15 operations application: A class of administrative application that provides an interface for the staff who operate a media library. Examples of operations applications include a program that directs operations staff to obtain a cartridge and load it into a drive, and a program that allows an operator to place a drive or robot in or out of service.

3.1.16 partition: A portion of a side of a cartridge that is accessible as a unit.

3.1.17 port: A physical entity that allows import or export of one or more cartridges from a library.

3.1.18 physical cartridge layer (PCL): *See:* **external layer.**

3.1.19 side: The physical portion of a cartridge that is accessible by a drive after one mount operation. A side contains one or more partitions.

3.1.20 vault: A non-automated (manual) library, i.e., a shelf.

3.1.21 volume: A named, logical container of data. A simple volume, which is the common case, is represented as a single partition. A complex volume utilizes multiple partitions to store its data; these partitions may be on one or more physical cartridges, which may be arranged in a number of different ways.

3.2 Acronyms and abbreviations

3.2.1 DCE: Formerly the Open Software Foundation (OSF) Distributed Computing Environment (DCE), and now the Open Group's DCE.

NOTE—See <http://www.opengroup.org> for technical specifications and references.

3.2.2 PCL: Physical cartridge label. *Syn:* **external label.**

3.2.3 UUID: Universally unique identifier. Various versions of UUIDs exist, and references for generation of UUIDs abound.

NOTE—See <http://www.opengroup.org> for technical specifications and references.

4. Functionality

The MMS manages media at the request of clients. MMS clients are application programs that run on one or more host computer systems. The MMS itself does not understand individual human users—it is the responsibility of the application programs that “own” the media to provide an appropriate mapping between the application's users and the data that an individual user may access on a piece of media. Since the MMS has no knowledge of users, the application must also know how to verify the identity of a particular individual user if user verification is required. In many cases, the application may utilize an operating system to handle user verification.

The reader may wish to consider the functions of the MMS from a number of perspectives.

- a) An individual, human user of physical media.
- b) A program or group of programs that manage and/or store and retrieve information on physical media.
- c) A site administrator or administrators who have policy responsibility for some group of media.
- d) Operations staff who must deal with individual requests, actions, and problem situations.

From the perspective of an individual human user, there should be little or no direct contact with the MMS. Applications may allow a user to gain direct access to media, such as mounting a tape archive (tar) on a POSIX or UNIX system. The user would then use the tar program to directly read or write the media. The MMS is structured to make it easy for these types of applications to exist, and for these applications to readily provide a degree of security consistent with the security model of the underlying computer system(s).

Client application programs have a relatively rich set of operations provided to them by the MMS. In addition to basic operations such as media allocation, mounting, dismounting, and deallocation, the MMS includes functions that allow a client application to store metadata along with the online control information that is maintained for each piece of media. Database-style operations are provided that permit an application to locate a specific piece or pieces of media based on standard MMS-provided criteria, as well as application-specific criteria. The MMS automatically maintains a moderate amount of metadata for each piece of media, and most of this information is available to application programs.

Administration and operation functions for the MMS are handled by a set of applications, the number and scope of which may grow over time. Core functions and levels of security are provided by the MMS architecture, but specific user interfaces are not constrained by the MMS suite. This permits the implementation of interfaces that are appropriate for different environments: a single user, a corporate data center, a television broadcast operation, a dairy farm, etc.

The MMS is engineered primarily to handle automated robotic libraries. It may also be used to manage vaults (manual libraries). There may be specific operational considerations with manual libraries that are not addressed optimally by the present version of the MMS.

4.1 Interface protocols

The interfaces in the MMS are application-layer protocols as designated in the ISO model. They are defined in terms of *interface languages*, which are text oriented and have an asynchronous command-response style. These interface languages are referred to as *protocols*, as they directly correspond to common application-layer Internet protocols such as HyperText Transfer Protocol (http) and File Transfer Protocol (ftp). To comply with an MMS interface, two components shall establish a connection via a network interface, and then send and receive information in one of the MMS-defined languages. (All currently defined MMS languages utilize text strings as the basis for information exchange.) TCP/IP shall be used as the transport for MMS protocols whenever information is transferred between MMS components. The standard port numbers 651, (ieee-mms) and 695 (ieee-mms-ssl) are assigned to the MMS by the Internet Assigned Numbering Authority and one or the other port number, or both, are used for all connections between components of the MMS.

This text-oriented metaphor was chosen for MMS instead of a binary interface such as DCE RPC, because the text-oriented protocol is more platform-neutral. The text-oriented message scheme also does not require the licensing, implementation, and installation of potentially complex underlying software. Implementers of the MMS may provide standard library code, which make it as easy to utilize text-oriented interfaces as an RPC system. (IEEE P1244.8 will provide a standard definition of these interfaces.) The functionality provided by the MMS interface languages is comprehensive and flexible, similar to a database query language such as SQL.

The ISO-10646 character set (Unicode) encoded in UTF-8 is used as the character set. The use of the ISO-10646 character set and other key parts of this standard ensure that systems that implement the IEEE MMS may be appropriately localized for use in different countries. All keywords are standardized and are not localized. Error and status messages are provided in a form that may be easily localized. Sort-order comparison of strings (as opposed to equality comparison) is performed in very limited cases, and localization support for collating sequence is provided in these cases.

There are four primary protocols defined in the MMS architecture for managing media. They are as follows:

- a) *Session Security, Authentication, and Initialization Protocol* (SSAIP)—IEEE P1244.2/D041900—is the initial *handshake* protocol used by all components of the MMS to establish identity, authority, and initial communication with the MM.
- b) *Media Management Protocol* (MMP)—IEEE Std 1244.3-2000—is used by client and administrative applications to allocate, mount, dismount, and deallocate volumes, and to administer the system. The MMP includes levels of privilege so that, for example, a client application cannot perform administrative functions, or an operator console program cannot perform higher-level management functions.
- c) *Drive Management Protocol* (DMP)—IEEE Std 1244.4-2000—defines the language by which the MM and DM communicate. The MM issues load and unload commands to the DM. The DM reports status, errors, usage statistics, and offered capabilities of the drive to the MM.
- d) *Library Management Protocol* (LMP)—IEEE Std 1244.5-2000—defines the language by which the MM and an LM communicate. An LM manages an automated library or a vault. The MM issues commands to the LM to move cartridges. The LM reports status, errors, and the current contents and topology of the library to the MM.

The MMP, DMP, and LMP are peer-to-peer, bi-directional, asynchronous protocols.

The minimum functionality required to implement an MMS is the SSAIP and MMP. (This would be a *monolithic* implementation that directly manages drives and libraries without using separate LM or DM programs.) A practical implementation will additionally include the DMP and LMP.

The following additional protocols are defined to extend the MMS to interoperate with other MMSs:

- a) *Media Manager Interchange Protocol* (MMIP)—IEEE P1244.6—will define a protocol to allow interchange of information between autonomous MMSs. This protocol will facilitate the transfer of media between separate MMSs by transferring all associated metadata that is associated with that media. The separate MMSs might be present at several sites within a large organization, or they might be in different organizations.
- b) *Media Manager Control Interface Protocol* (MMCIP)—IEEE P1244.7—will define a protocol that permits interfacing the data management component of the MMS with existing library management systems. The MMCIP will allow real-time coordination of metadata information between an MMS and another non-IEEE 1244 MMS, whereas the MMIP will allow off-line *shipment* of metadata between two instantiations of MMS.

4.2 Programming and command line interfaces

In addition to the protocols that specify the functionality of the MMS, programming and command-line interfaces are defined as follows:

- a) *C Language Procedural Interface*—IEEE P1244.8—will define a set of standard programming interfaces that facilitate the construction of components of the MMS, particularly client, administrative, and operational applications, LMs, and DMs. (The initial definition will be for the C programming language; additional languages such as C++ or Java could be added as additional standards or as an addition to IEEE P1244.8 at a later date.)

- b) *MMS User Mount Commands*—IEEE P1244.9—will define a set of standard commands to allow a user to allocate, mount, unmount, and deallocate media. These commands are specified as a part of a command line interface for systems that offer these interfaces, such as the POSIX or UNIX shell, or Windows NT[®] command-line interface.⁷ Commands may be embedded in scripts to produce more complex or custom functions, or to allow an application program that is not written for an MMS to be adapted for use with an MMS.
- c) *MMS Standard Administrative and Operational Commands*—IEEE P1244.10—will define a set of standard administration and operation commands of an MMS. The standard defines a command-line, minimally interactive interface for basic interaction with the MMS; these commands could be used to construct interactive interfaces using scripting-based systems such as shell scripts, Web CGI scripting, Perl, or tcl/tk.

An implementation of the MMS does not require any of these interfaces.

4.3 Data transfer protocol and interface

This standard expects that input/output (I/O) between applications and devices will be performed locally on a machine, using that machine's host operating system primitives for I/O. Access control and protection for the drive, and the media loaded in the drive, is controlled by the host operating system.

- a) *Media Data Mover*—IEEE P1244.11—defines protocols and programming interfaces to allow remote and protected access to drives. It permits local and remote input-output operations to be performed in a portable, system-independent manner, and implements the protection of label information on media. A data mover is not required to implement the basic functions of an MMS, nor is it required for operation of an MMS.

4.4 Operation of the MMS

The MMS is designed to provide a high degree of independence between modules, allowing parallel development by multiple organizations of these modules, and enabling an end user to *mix and match* modules written by different organizations. The standard interface protocols described in 4.1, 4.2, and 4.3 are, of course, key to this characteristic. The primary modules of the MMS are depicted in Figure 1.

The MM is the central program module of the MMS. For a given installation, there is a single instantiation of an MM that manages all the media for that site. If separately administered sets of media exist within an organization, multiple MMs could be used. The MM includes a persistent store, which holds all relevant information about the site, including all media, drives, hosts, connectivity between hosts and drives, compatibility between media and drives, applications, etc. The MM receives requests from applications, and utilizes its knowledge of the state of the site to perform resource allocation and operational actions.

The architecture and protocols of the MMS are designed to accommodate failures of system components. The MM is accessed by other components of the system using a known TCP/IP address. In a high-availability configuration, persistent information stored by the MM must be preserved and made available to the entity that takes over the known TCP/IP address after failure. Address take-over can be done using standard Internet Protocol (IP) fail-over technology such as IBM's HACMP[™] or Silicon Graphics' IRIS Fail-Safe[™]. The fail-over mechanism is not defined by this standard, nor is it required to be present. The only requirement is that the MM always have the same IP address.

⁷The first time a trademarked product is listed in this standard, it will be marked with either a ® or TM. Subsequent referrals to the trademark item will be capitalized only. Please refer to the front matter of this standard for details on trademarks.

One or more LMs and DMs carry out the actions requested by the MM. To move a cartridge to a drive, the MM utilizes the LMP, communicating with the appropriate LM. The LM performs whatever library-specific actions are required to cause the library to move the cartridge into the selected drive. Under certain circumstances, such as a tape stored in a vault, which must be read by a drive that is attached to an automated library, the MM may need to communicate with more than one LM in order to move the selected cartridge into the selected drive.

Once the cartridge has been loaded into the drive, the MM performs any necessary control operations on the drive via the DMP to the DM associated with the drive. As with the LM, the MM does not have knowledge of the specifics of the drive; the DMP provides a generic interface, and the DM specific to the selected drive performs any drive-dependent functions. Optionally, the MM may also verify the internal label(s) on the media, performing any necessary I/O via the DMP and the DM.

The intent is that the LM and DM be relatively simple programs, and the majority of complexity of the MMS be implemented in the MM. Typically, LM and DM components can be implemented so that any required dynamic state can either be obtained from the device or can be maintained in private persistent storage areas within the MM. This allows rapid restart as well as the implementation of high-availability configurations of LMs and DMs.

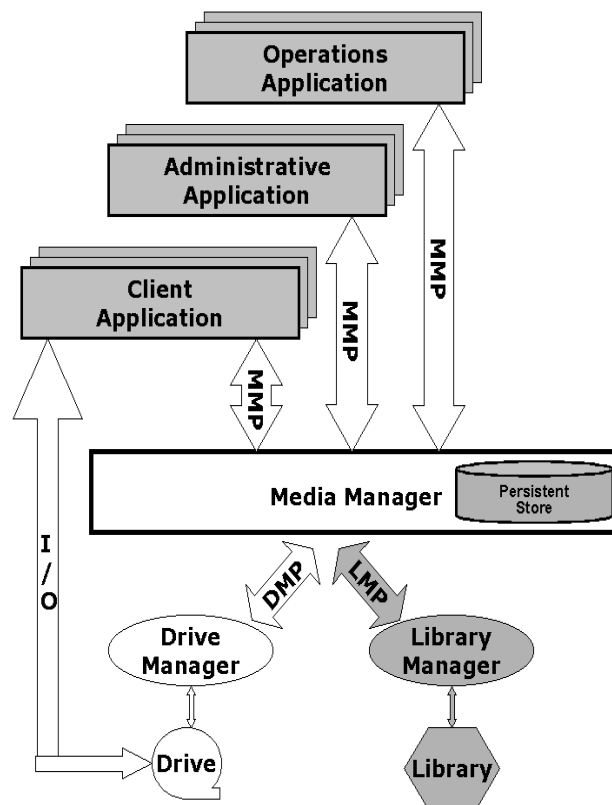


Figure 1—Primary modules within the MMS—the Media Manager, Library Managers, and Drive Managers

4.4.1 Operational overview

To put the system components and protocols into perspective, here is a relatively simple example that illustrates how the pieces of the system fit together. In this example, the sequence of events is as follows:

- a) An application program starts up.
- b) The application connects to the MM and is authenticated.
- c) The application causes the MMS to mount a tape.
- d) The application performs I/O directly to the tape.
- e) The application utilizes the MMS to dismount the tape.
- f) The application exits.

Communication between the application and the MM is shown as well as communication between the MM and the DMs and LMs.

This example is not an authoritative description of each of the languages that are used within the example; consult the appropriate individual standard for the language for an authoritative description.

4.4.1.1 Application establishes connection with MM

The application utilizes its network configuration information to establish a TCP/IP connection to the MMS on the system that is running the MM. Once the connection is established, it remains open until the application is completely done with its interaction with the MM. After the TCP/IP connection is established, the application *signs on* to the MM using the SSAIP. (For details of the SSAIP, see IEEE P1244.2/D041900.) This sign on process is slightly more complex than a typical interactive log-on because it establishes the identity of both parties, the level of security to be used for the remainder of the connection, and the language to be used for communication during the session. If the application does not need encryption of the session, it should open TCP port 651 on the MM's host, while if it needs secure socket layer (SSL) encryption, it should open TCP port 652 on the MM's host.

```
Application -> MM:      hello client ["Backup"] instance ["Accounting"]  
                        language ["MMP"] versions ["1.0" "1.1"]  
                        password["cablefree"];
```

In the example above, the application identifies itself to the server as the *Accounting* instance of the *Backup* application. The application then states that it will be utilizing the MMP language, and is capable of speaking either version 1.0 or 1.1 of that language. It also establishes an authenticated session using simple password authentication. If the application did not require authentication, the password clause would have been eliminated; see the IEEE P1244.2/D041900 for details.

When the MM receives this message, it retrieves its copy of the private shared key specific to the *Accounting* instance of the *Backup* application. If the passwords do not match, the client is refused access. If the passwords match, the application is authenticated, and the MM replies with its name.

The MM replies with the version of the language that will be utilized for this session and a password that the application can use to verify that it is talking to the real MM process.

```
MM -> Application:      welcome version["1.1"] password["silverton"];
```

All further communication on the TCP/IP connection between the application and the MM is in the MMP language.

Applications typically interact with only a single MM, but the SSAIP protocol is designed to permit an application to interact with more than one MM. If the authentication information does not match, the application should terminate the session. If the authentications do match, the MM has authenticated itself to the application.

4.4.1.2 Application requests a volume to be mounted

The application sends a simple *mount* command to the MM, which requests that the volume *usr990424* be mounted.

```
Application -> MM: mount task ["1"] volname ["usr990424"]
                    report [MOUNTLOGICAL."MountLogicalHandle"];
```

The *task* clause identifies this command and is used to correlate any responses from the MM with this specific request. The *report* clause instructs the MM to return the logical device handle on which the volume is ultimately mounted; without this information, the application would have no idea how to access the media. The MM would typically *accept* this command immediately, so that the application could, if appropriate, issue further commands to the MM for processing while the following command executes:

```
MM -> Application:    response task ["1"] accepted;
```

The command-accept-response model is described in 7.3.

The MM then consults its database to determine

- a) Whether or not this volume exists in the application's name space.
- b) The specific physical media (cartridge, side, and partition) to which the volume refers.
- c) In which library the requested cartridge resides, and its location within the library.
- d) Which drives within that library could be used to mount the cartridge, considering the cartridge's type and the physical connectivity of drives to host computers.

If either the cartridge is in use, or all possible drives on which the cartridge could be mounted are in use, the request will be delayed until appropriate resources become available. Once those resources are available, the MM commences the physical task of getting the cartridge into the drive and making it ready for use.

4.4.1.3 MM interacts with LM to mount physical media

The MM is connected to the selected LM via a TCP/IP connection. This connection was established when the LM and MM were brought up, and remains established until one or both of these system components terminate. An initialization exchange similar to the one shown in 4.4.1.1 is used to establish the application-level connection, authentication protocol, and language version. The language used between the MM and the LM is LMP/M; commands from the LM to the MM are in the LMP/L language.

In this example, the cartridge is identified by the physical cartridge label *AB1234* and is to be mounted on side 1. It currently resides in *bay 2, slot 12* of the library, and is to be mounted in either drive *drive1* or drive *drive2*, at the LM's option.

```
MM -> LM:          mount task ["365"]
                    slot ["bay 2, slot 12" "AB1234" "side 1"]
                    drive ["drive1"] drive["drive2"];
```

The LM accepts the command as follows:

```
LM -> MM:          response task ["365"] accepted;
```

The LM then causes the library to perform the physical action of moving the cartridge from the slot to the drive and mounting it in the drive. (The MM maps the slot name in the MMP *slot* command to an associated bay name for the LM; both slot names and bay names are arbitrary strings, but implementers are strongly encouraged to choose names that have meaning to human operators.) Once this action has been successfully completed, the LM updates the state of several internal tables in the MM.

```
LM -> MM:      config task ["982"] scope ["partial"]
                slot ["bay 2, slot 12" "bay 2" "AB1234" "DLT"
                    "occupied" "accessible"]
                bay["bay 1" "DLT" "0"]
                freeslots["bay 1" "DLT" "0"]
                drive ["drive1" "bay 1" "AB1234" "occupied"
                    "accessible"];
```

(The MM's *accepted* and *success* responses to the *config* commands are omitted here.)

Finally, the LM tells the MM that the requested cartridge was mounted in drive *DLT202*.

```
LM -> MM:      response task ["365"] success
                text ["bay 2, slot 12" "AB1234" "drive1"];
```

4.4.1.4 MM interacts with DM to set up drive

The MM is connected to the selected DM via a TCP/IP connection. This connection was established when the DM and MM were brought up, and remains established until one or both of these system components terminate. An initialization exchange similar to the one shown in 4.4.1.1 is used to establish the application-level connection, authentication protocol, and language version. The language used between the MM and the DM is DMP/M; commands from the DM to the MM are in the DMP/D language.

The DM is located on the same physical computer as the application, and utilizes the same physical connection to the drive that the application will use.

The MM requests that the DM ensure that the cartridge is loaded in the drive:

```
MM -> DM:      loadtask ["191"];
```

The DM accepts the command:

```
DM -> MM:      response task ["191"] accepted;
```

The DM causes any physical activity that might be required to load the drive, and then the DM reports that the command has completed successfully.

```
DM -> MM:      response task ["191"] success;
```

The MM requests that the DM make this drive available for read-write access, and return to the MM an appropriate device name to pass back to the application.

```
MM -> DM:      attach task["192"] modename ["readwrite"];
```

Readwrite is a standard token; the base set of standard tokens appears as Clause 20 of this standard. The DM accepts the command:

```
DM -> MM:           response task ["192"] accepted;
```

The DM creates any necessary device nodes, conditions the hardware and/or operating system software for the mode requested in the *attach* command, and then replies to the MM with the software name of the device that the application will use to access the cartridge.

```
DM -> MM:           response task ["192"] success
                    text ["/dev/mmtape/8E927BD4"];
```

In this example, the DM created a *one-time-only* node named */dev/mmtape/8E927BD4* for access to the device.

4.4.1.5 MM replies to application

The MM replies to the application that the requested volume has been mounted on the node */dev/mmtape/8E927BD4*:

```
MM -> Application:  response task ["1"] success
                    text ["/dev/mmtape/8E927BD4"];
```

This completes the execution of the application's *task 1* command.

4.4.1.6 Application performs I/O to drive

The application may now perform I/O to the drive, reading and writing the media as it wishes. All I/O is performed through standard device mechanisms within the operating system on which the application is running.

4.4.1.7 Application requests volume to be unmounted

The application sends an *unmount* command to the MM, which requests that the volume *usr990424* be unmounted.

```
Application - > MM:  unmount task ["2"] volname ["usr990424"];
```

The MM accepts this command:

```
MM -> Application:  response task ["2"] accepted;
```

The MM then interacts with the DM and MM to physically unmount the cartridge; these interactions are similar to those shown in 4.4.1.3 and 4.4.1.4, and are omitted here for brevity. Once the physical actions are complete, the MM informs the application:

```
MM -> Application:  response task ["2"] success;
```

4.4.1.8 Application terminates connection with MM

The application is free to continue interacting with the MM, performing other mounts, allocation, unmounts, etc., until it is finished. The application terminates its interaction with the MM by issuing the *goodbye* command:

```
Application -> MM:      goodbye task ["3"];
```

The MM accepts this command:

```
MM -> Application:      response task ["3"] accepted;
```

If there are any pending actions that the MM must complete before the session can be terminated, there will be a delay before the MM's final reply to the application:

```
MM -> Application:      response task ["3"] success;
```

The application then closes its TCP/IP connection to the MM.

Note that the application could also terminate (and indeed, cause the *unmount* to occur) by unilaterally closing the TCP/IP connection (this happens when an application crashes, for example). However, simply dropping the connection doesn't give the MM the opportunity to gracefully complete any commands that might be outstanding, nor to return status information to the application. See 6.2.6 for more details on abnormal termination.

4.5 LM operational overview

Every LM implements the LMP, which allows a library to be controlled by the MM in a standard manner. The LM may be implemented on a host system that controls the library via a local peripheral attachment (serial, SCSI, etc.) or could be implemented in an embedded controller that is a part of the library. A library with an embedded controller would simply plug into the local area network and immediately be usable as part of an MMS. (We would expect that any required configuration of such a library would be performed by a few Web pages.)

Libraries vary widely in their capabilities, functions, and interfaces, so LM implementations will differ from library to library. There are three broad categories of libraries, each of which may require a somewhat different implementation strategy. They are as follows:

- a) Robotic libraries with bar-code readers or other means of externally verifying the identity of a cartridge (*sighted robots*)
- b) Robotic libraries without bar-code readers or other means of externally verifying the identity of a cartridge (*blind robots*)
- c) Libraries that are operated by human beings (*vaults*)

In all cases, the most authoritative representation of the state of the library is the physical world. State includes such information as the physical configuration of the library, the position within the library of each cartridge or open slot, drives in the library, etc. The MM, with the cooperation of the LM, maintains a representation of the state of the library in its databases; see Clause 13 for details on the MMs representation of a library. The *config* command in LMP is used to update the MMs databases whenever the state of the library changes. The implementers of an LM may choose to maintain no state locally within the LM and simply operate strictly using information contained in the commands that are received by the LM from the MM, or may elect to maintain a temporary copy of the state of the library.

One key difference between sighted robots and both blind robots and vaults is the ease with which an inventory of the library may be obtained. For many sighted robots, it is a routine procedure to have the library physically inventory the contents of the library whenever the unit is powered up, or whenever the unit has been opened in such a way that the contents might have been disturbed. For a blind robot, the MMS must rely on its stored database, as the library cannot tell the MMS the identity and location of individual

cartridges. (Some blind robots can determine whether or not a slot is empty.) A manual inventory can be done in a vault, but it is potentially a time-consuming operation. Some installations may be able to use hand-held bar-code readers to speed this process, but a manual inventory is still not something that would be carried out frequently.

Vaults require, and robots may require, interaction with a system operator. The MMS provides a mechanism, the *request* command in the LMP, by which the LM may utilize the MM to interact with human operators. The MM, in turn, utilizes one or more administrative applications and the operator interface commands in MMP (*accept*, *respond*, *release*, *move*, *eject*, and *inject*) to perform those interactions with the operator(s). (An LM should not directly interact with an operator console, because in larger installations, it may be desirable for one or more libraries to share one or more operator consoles.)

4.5.1 Mount verification

A key issue with managing removable media is to ensure that the desired piece of media actually gets mounted in the desired drive. With a sighted robot, the probability that the operation is performed correctly is essentially 100%. With a blind robot, the probability of an error is higher, because the robot cannot verify that the desired cartridge (actually, the cartridge containing the desired partition) is actually present in the slot that the database says the cartridge is in. With a vault, an operator will typically visually verify that the right cartridge and partition are obtained, but operators do sometimes make mistakes.

The MMS is system independent. Since a number of popular operating systems such as UNIX[®] do not have any notion of standard tape labels, the MMS cannot simply require that all cartridges have standard machine-readable labels (such as ANSI or IBM[®] standard labels). Without standardized tape labels, the MMS cannot have the DM verify that label at mount time. Instead, the MMS provides an extensible mount-time verification scheme.

Verification may be performed in one or more ways, listed as follows in order of precedence:

- a) If the library is a sighted robot with bar-code validation capability, it verifies the bar code on the cartridge as the cartridge is being mounted. If the bar code does not match the expected bar code, the LM will cause the mount to fail. If the bar code is verified, then the on-media identifier [step b)] is checked (if that capability is available on the drive); otherwise, no further checking is performed.
- b) If the cartridge and drive support an on-media identifier, the DM will supply that identifier to the MM as part of the response to the load command. (The on-media identifier is a device-dependent immutable identifier, which is automatically encoded on the media by the drive.) The on-media identifier will be compared with the expected media ID that is stored in the MMs database in the PARTITION.PartitionSignature field. This check is performed for all mounts, even those for which bar-code verification is available, if it is available on the cartridge and drive. If the media ID does not match the one stored in the MMs database, the mount is aborted by the MM.
- c) If the application has been administratively permitted to bypass the signature mechanism [step d)] and the application requests no verification at mount time, no further checking is performed. It is assumed that the application will check to ensure that the correct volume has been loaded, and that the application will reject the volume if it is not the one that was expected.
- d) A *signature* mechanism is used to do a best efforts check on the cartridge.

The signature is a short representation of the content of the cartridge—this may be a hash sum of the first megabyte of data on the cartridge or a tape label, several forms of which are possible. Under certain rare circumstances, an application-specific label may be used to identify a cartridge, such as when the label information is at the end of the media. The signature and signature type for each cartridge is stored in the MMs database. When a cartridge is mounted, MM uses the DMP's *identify* command, which includes the

type of signature being requested, to cause the DM to return the cartridge's signature after *loading* the cartridge and *attaching* the device.

When a cartridge is unmounted, the MM uses the *identify* command to obtain from the DM an updated signature for the cartridge prior to *detaching* and *unloading* the cartridge. This additional *identify* step can be bypassed for applications by identifying the unmount as *clean*. An application should do so only if it has either supplied the new signature directly to the MM by directly setting the PARTITION.PartitionSignature attribute or if no change to the signature portion of the partition was made by the application.

Further details on the mount-time media verification and signature mechanism are given in the *identify* command description in the DMP.

4.5.2 Introduction of media

This subclause outlines how an operator may introduce media to a library and how the MMS handles this operation. The operator may choose to physically place the media in the library and then work with the MMS to appropriately identify the media, or the operator may choose to pre-identify media which the operator will then introduce into the library.

The operator interface is an administrative application, which utilizes MMP's *inject* command to instruct the MM to take one or more cartridges into the library. The MM utilizes the LMP's *inject* command to cause the library to perform the physical operation. Detailed information on the command sequence is presented in IEEE Std 1244.3-2000.

4.6 System start-up, restart, and shutdown

Components of the system may be started in any order, but no processing will occur until the MM is in operation. When started, all applications, LMs, and DMs attempt to contact the MM at a known network address (typically specified using a host name that is looked up via DNS to yield an IP address) on the assigned port number 651 (*ieee_mms*). If no MM is available, the application, LM, or DM will sleep for some period of time and try again. If the MM is available, the SSAIP (IEEE P1244.2/D041900) is used to establish and authenticate the identities of the two parties, and to resolve the protocol that is to be used for further communications.

In the event of unexpected termination of any system component, the system component with which that component is communicating will be notified of the failure of that component by the networking subsystem. If the failed component is an application, LM, or DM, the MM will take appropriate action(s) to clean up any items that are in process. When the failed component is restarted, it will establish communication with the MM as described above. Resynchronization of state with the MM is left up to the application, LM, or DM.

If the MM should terminate unexpectedly, or should a network partition occur, all application, LM, and DM components should reset themselves and attempt to re-establish a new connection with the MM. The MM may be restarted on the same or a different machine, as long as its static databases and its IP address are preserved. Autonomous operation of applications, LMs, and DMs is prohibited.

More detailed information on start-up, termination, and recovery from failures is given in Clause 6 of this standard and in the individual language specifications for the MMP, DMP, and LMP.

5. Operation and matching model

Each statement in the MMP, DMP, and LMP languages consists of a command keyword followed by appropriate arguments. The command specifies an action, and the arguments express options or *constraints* on

how the command proceeds. For example, the *mount* command can specify the volume to be mounted explicitly by name, or by using a form of Boolean algebra to determine one or more candidate volumes to be mounted. Thus, the MM may be thought of as a *solution engine*, i.e., the system fulfills requests by finding good solutions to the declarative requests. This is true as well with the LM—the MM may make either totally specific requests or reasonably general requests to the LM. A general request allows the MM to make optimal choices using information that is specific to the make and model of library, and even with respect to the library's current state.

For applications, the Boolean expressions may operate not only on standard-defined metadata information that is stored in the MM's database, but also on application-defined information, such as the approximate percentage of a volume that is in use.

As an example, consider the *match*, *order*, and *number* clauses that are common to a number of commands in the MMP. The *match* clause takes a Boolean expression that selects one or more cartridges within the MM's database. The *order* clause sorts the results of the *match*, and the *number* clause selects particular records out of the results. Thus, using a *match/order/number* mechanism and an application-defined *percent in use* field, an application could select the three *emptiest* volumes of a specific cartridge type that belong to an application and cause those volumes to be mounted.

6. System model

A system that conforms to a standard IEEE MM may be used in a wide range of installations, from very small to global enterprises. An example of a small installation is a single PC on a desktop with a single removable-media drive, and a desk drawer containing a few cartridges. There is no system administrator – the user of the system relies on the MMS to keep track of the cartridges and to work with applications that use MMS services to request that the user mount or dismount a cartridge. An MM used by a global enterprise could include thousands of automated libraries and vaults, tens of thousands of drives and computer systems, and tens of millions of cartridges. This MMS would be administered by full-time professional staff, communicate with dozens of human operators across a number of sites, and interact with hundreds of distinct applications and thousands of users.

The protocols used to communicate between the various components of the MMS are the same, regardless of the size of the installation. An installation will contain the same basic software components, regardless of its size, but the number of components and the implementation of these components is likely to differ depending on the size of the installation.

This clause describes the software and hardware components of the MMS. Clause 7 gives an overview of the protocols used for communication between the components; detailed specifications of the individual protocols are given in other standards in the IEEE 1244.x family. This standard describes the data model that underlies the system as well.

6.1 Media

The purpose of the MMS is to manage the physical media associated with an installation. Systems managed by the MMS will support a diverse range of removable media, including computer media such as magnetic tape, optical disk, and CD-ROM, as well as non-computer media such as audiotape and videotape, film, and audio CDs and videodiscs.

To properly manage media, the MMS must understand a moderate amount of information about a physical piece of media, henceforth referred to as a *cartridge*. This information includes the following:

- a) Physical characteristics, such as its external, physical form factor; the type of media housed inside the external shell; the method used to record information on the cartridge; any physical identification present on the outside of the cartridge.
- b) Usage and life cycle information, such as the number of mounts, dates first and last written, expected longevity.
- c) The location where the cartridge is normally stored, its current location if it is not in its normal location.

Many types of media can accommodate more than a single logical entity within a physical cartridge container. They are as follows:

- Optical or magnetic disks may have more than one *side*, and a drive that handles media with multiple sides may allow access to only one or to all sides simultaneously.
- Optical or magnetic disks may have more than one *partition*; depending on the device, it may be practical to access multiple partitions simultaneously.
- Magnetic tapes may have more than one partition; typically, only one partition may be accessed at a time.

Sometimes, a user may wish to refer to a group of cartridges as a single entity, for example when using striped or other forms of redundant tape. This group of cartridges is represented as a *cartridge set*. Note that cartridge sets are not supported by the initial MMS standard. Also note that cartridge sets are not to be confused with *cartridge groups*, which are a separate object type in the data model.

The notion of a *volume* in the MMS allows users to think in terms of a logical unit of information that corresponds to a single physical cartridge, a partition within a cartridge, or a group of cartridges. In almost all cases, applications and end users refer to MMS volumes. The MMS maps the named volume onto an appropriate cartridge or cartridges when performing physical operations.

6.1.1 Cartridges

Cartridges are the physical media that the MMS manages. A cartridge has an external shape that determines the slots of a library in which it can reside, and the drives in which the cartridge can physically fit. The media within the cartridge and the bit format that is, or will be, recorded on the cartridge also determine the type(s) of drive in which the cartridge may be used.

Some cartridges have additional characteristics that are of importance to the MMS, including partitions and sides. Some optical cartridges and DVDs have sides—they can be mounted with one side or the other accessible, and the other side inaccessible. The two sides can be treated as independent pieces of media except that they cannot be mounted simultaneously. Also, if a two-sided piece of media is physically removed and transported, the two sides are bonded together.

Ampex[®] DST[®] and some other tape media may be partitioned into multiple separate *tapes*, each of which has a pre-determined size. An application that has access to a drive in which a partitioned tape is loaded would be able to read and write only that partition, and would encounter an end-of-file if an attempt were made to write past the end of the partition. Similarly, many removable disk media, both magnetic and optical, may be partitioned into distinct partitions that have protected access characteristics. The MMS allocates volumes (the logical entity that applications actually use) to partitions rather than to cartridges.

Disk devices, including optical disks, are frequently used by multiple users at the same time—a file system partition, for example, might reside on an MMS-managed volume on an optical disk. This version of the MMS architecture does not directly support multiple MMS client applications that simultaneously access a single cartridge. To use shared-access media with this version of the MMS, an intermediary application is

required that acts as either a multiplexer or as a mounting service. For example, a magneto-optical disk might contain a filesystem that could be mounted by an application. The mounting application would use MMS interfaces to mount the volume on which the filesystem resides, and then mount the filesystem; once the disk is mounted, other applications would use standard filesystem interfaces to access the media, not MMS interfaces. The mounting application would need to remain active until the filesystem was unmounted, at which point the mounting application would unmount the volume. (Note that the mounting application shall not terminate while the filesystem is in use, as the MMS would then automatically reclaim the drive and the media.)

The current MMS does not allow simultaneous access to multiple partitions on a single cartridge, even if the underlying hardware would permit this operation.

A cartridge has a number of characteristics that describe the cartridge and allow the MMS to identify the cartridge and to determine where the cartridge may be mounted.

- a) A *physical cartridge label* (PCL) plus the external cartridge shape (form factor or slot type) of a cartridge uniquely identify each cartridge in the system.
- b) A *unique ID* (UUID) is used internally within the MMS to identify the cartridge.
- c) The *external cartridge shape, media oxide type, bit format*, and partition and side information are used to determine conformability of media with drives.

Statistical information such as number of mounts, error counts, is also maintained with each cartridge, which facilitates management of the life cycle of the cartridge.

Detailed information on the data model for cartridges is in Clause 15 of this standard.

6.1.2 Volumes

The volume is the unit of logical information that an application manipulates with the MMS. In the case of a simple cartridge (one side, one partition), there is a one-to-one correspondence between the volume that the application views and the physical cartridge.

Volumes are named by applications using strings of the application's choice. Volume names are unique within an application: different applications may choose to have volumes with identical names. Applications have no visibility to volumes that are owned by other applications. However, an administrative application may, through the use of references that are qualified with application names, obtain access to all volumes within the MMS.

The application is the only level of access control provided by MMS for volumes—any finer-grained protection, such as on a per-user basis, is expected to be provided by applications.

6.1.3 Cartridge life cycle

The life cycle of a cartridge is the sequence of states and events that describe the history of a physical cartridge from the time that it first becomes part of the MMS until it ceases to be a part of it. The Media Manager Interface Protocol, (IEEE P1244.6), defines a mechanism by which the history of a physical cartridge can be transferred between distinct IEEE MMS installations when the physical cartridge is transferred between installations.

The major events in the life of a cartridge include the following:

- a) The physical and logical introduction of the cartridge into the system

- b) Assignment of ownership (who or what application gets to use the cartridge)
- c) Use of the cartridge by applications
- d) Recycling of a cartridge when one owner no longer needs it
- e) Disposal of the cartridge, either by sending it to another system, or by removing it for disposal when the media reaches the end of its service life

Certain information on a cartridge is maintained by the MM during its lifetime; other information may be added to the data stored by the MM by administrative applications. These administrative applications may also utilize the information that is automatically maintained by the MM—for example, this information would allow a life cycle management application to locate cartridges that are in need of replacement either due to error statistics or due to number of uses.

The MM utilizes a state model to describe its view of the life cycle of a cartridge. First, a cartridge must be introduced into the MMS:

- 1) If a cartridge is introduced into a library (via an MMP *inject* command) and is otherwise unknown to the MM, it is in the *located* state.
- 2) If a cartridge has been *defined* by an administrative application to the MM, but has not yet been seen by the MM in any managed library, it is in the *defined* state. A cartridge in the *defined* state optionally may be associated with an application.
- 3) Once a *located* cartridge has information added by the administrative application, or a *defined* cartridge is physically introduced into the system, it becomes an *identified* cartridge or an *allocatable* cartridge. A cartridge that is *identified* is not associated with a cartridge group; an *allocatable* cartridge has been assigned to a particular cartridge group.
- 4) An *identified* cartridge becomes *allocatable* when it is assigned to a particular cartridge group. This may happen via an administrative application, or when an application requests allocation of an otherwise unassigned cartridge.
- 5) When an application allocates the cartridge and associates a volume name with it (the volume name is actually associated with a partition within a side of a cartridge), the cartridge moves to the *allocated* state.
- 6) When the application is finished with the volume (actually, finished with all allocated volumes on a cartridge), the cartridge becomes *deallocated*. The cartridge remains owned by the application until an administrative application processes it.

The administrative application that processes *deallocated* cartridges may implement site-specific policies, such as erasing headers, writing the entire media, causing it to be degaussed by an operator, or even physically destroying the media. When this application wishes to make a cartridge available for reuse, it causes the cartridge to enter the *recycled* state; this state is equivalent to *identified*. *Recycled* and *identified* are distinct to differentiate cartridges that are newly introduced into the system.

An *error* state is defined for a cartridge that temporarily cannot be used but should remain associated with an application. A cartridge may move from the *allocated* state to *error*, and from *error* to either *allocated* or *deallocated*.

Detailed information on the cartridge life cycle may be found in IEEE Std 1244.3-2000 (MMP).

6.2 Applications

The MMS is designed to work with one or more end-user applications that implement data I/O functions. It is possible to have an installation in which there are no I/O applications, or a system in which I/O applications are not computer programs; this would be a special case in which the MMS is used to manage non-computer media, such as video or audio tapes or discs. Administrative and operator interface applications are also responsible for providing all administrative and operational interfaces that may be required by an installation.

All applications communicate with the MMS using the MMP, as defined in IEEE Std 1244.3-2000. An overview of the MMP is provided in 7.7. There is a well-defined subset of the MMP that is used for non-privileged applications. The operations available in the subset are sufficient for applications engaged in common tasks such as data transfer and mounting user media. The additional operators in the full MMP are intended to enable the creation of *administrative applications*. This model defines two levels of privilege within applications that utilize the MMP: *application and administrative*; administrative applications have no restrictions on the operations they may perform, similar to the POSIX or UNIX super-user. Future versions of the MMS will allow multiple levels of privilege to be defined for MMP operations.

To aid development of applications using the MMP, the *C-Language Procedural Interface*, IEEE P1244.8, will define a standard C-language programming interface that simplifies and standardizes the use of the MMP. Application development in scripting languages such as *Perl* may be performed using the *MMS User Mount Commands*, IEEE P1244.9, and using the *MMS Standard Administrative and Operational Commands*, IEEE P1244.10. These two standards, when written, will define command-language interfaces for user mounting and administrative and operational commands. Specific interfaces for other programming languages may be defined in the future.

6.2.1 User applications

A typical application uses MMS services to allocate, mount, unmount, and deallocate volumes. Data transfer applications utilize volumes, which are logical containers of information, rather than cartridges, which are physical entities. For most current tape technologies, there is only one volume per cartridge, and that volume covers the entire user-accessible area of the cartridge.

A user mount application (UMA) is another example of a standard application, one that would normally be provided as a standard component of an MMS implementation. The UMA allows human users of a computer system to allocate, mount, unmount, and deallocate volumes. The UMA is necessary because the MMS is not aware of the security requirements, user identity conventions, or protection mechanisms of each specific type of computer system or operating system. The MMS itself is concerned with applications, and ensures that applications only have control over media associated with the application requesting the media.

IEEE P1244.9 (see 4.2 of this standard) defines a UMA interface that will be consistent across many different computer platforms.

6.2.2 Administrative applications

Applications that are intended to change the parameters or configuration of the MMS are called *administrative applications*. Administrative applications are given a level of privilege that is analogous to the *super-user* privilege in POSIX or UNIX systems. These applications can perform essentially all allowable operations within the MMS. A future version of this standard will define a more detailed model for selectively granting access to specific features and capabilities within the MMS.

Many functions that are necessary or helpful in a complete MMS are performed by administrative applications instead of by built-in, standardized components. Administrative applications span a wide range of needs, including the following:

- a) Operator interface for non-automated libraries (vaults)
- b) Operator interface for automated libraries
- c) Introduction and removal of media
- d) Changing the availability of hardware devices (drives and libraries, up and down, accessible to which machines, etc.)
- e) Monitoring status of devices and media
- f) Causing drive cleaning to occur
- g) Adding and deleting application access to the MMS
- h) Obtaining current system status
- i) Prioritizing and/or canceling individual commands and requests
- j) Performing inventory, particularly of a non-automated library (vault)

This list of typical administrative applications is not intended to be complete or comprehensive.

Certain unusual requirements could require that a data transfer application have the ability to perform a privileged operation, (one that is not normally accessible to a data transfer application). A site may grant administrative privilege to a data transfer application to permit this operation, but by doing so it would also grant full access to all privileged operations. The design of the operations allowed for non-privileged applications should make this case highly unlikely.

IEEE P1244.10 (see 4.2) will define an Administrative and Operational Command Interface that will be consistent across many different system platforms. This interface will be useful for implementing administrative tools using scripting languages.

6.2.3 Operator functions

The MMS includes facilities that permit integrated dispatch management of functions that are performed by human operators. LMs and DMs may utilize the *request* command of LMP/L or DMP/L to send a request to a human operator. Pending requests are held in a queue within the MM, and this queue may be examined and manipulated by one or more administrative applications. Utilizing privileged commands in MMP, an individual administrative application may *accept* responsibility for command or may *release* a request that it had previously *accepted*. Accept and release could either occur by program logic or by manual operator selection through an interface provided by the administrative application.

When the requested operation has been completed (or rejected), the administrative application uses the MMP's *respond* command to send a response to the requesting DM or LM.

One particular case of interest is a non-automated library (or *vault*). The LM for a vault receives an LMP *mount* requests and in turn generates LMP/L *request* commands for an operator to perform the necessary physical actions. An operator interface application removes requests from the pending queue, allows the operator to act on them, and then obtains a response from the operator. The operator interface application informs the LM of the action taken via the MMP *respond* command, and the LM can then complete (or refuse) the mount.

Although it is possible to write a vault LM that directly interacts with the operator, this is not recommended, as it would not allow integration of this LM with other LMs in a single operator interaction environment.

6.2.4 Configuration management

The MMS has considerable configuration flexibility built into the data model described in this standard. Network configuration information, which resides outside the data model, is used to establish a few basic configuration parameters. The most important of these are parameters for individual Applications, Libraries, and DMs. This network configuration information includes the following:

- a) The network name (or IP address) of the machine on which the MM resides
- b) The name of the MM
- c) The name of the Application, Library, or Drive
- d) The name of the Application Instance (AI), LM, or DM
- e) The private (secret) key that is used to authenticate the LM, DM, or application with the MM

This standard does not address the method by which this network configuration information is generated or distributed to the individual components.

Corresponding configuration information is stored in the MM's internal database.

One or more administrative applications with appropriate internal security controls are used to manipulate the configuration information that is stored in the MM's internal database. One key design goal of the MMS was to minimize the amount of *hard-wired* static knowledge in the MMS. This design goal is met by having a highly extensible set of data that is stored in the internal database. For example, new types of cartridges, drives, and libraries, each of which have characteristics that would not be known to the MMS beforehand, can be defined by an installation. Error and informational message texts may be added to the MMS database, both to accommodate new equipment and to introduce new languages to the system.

6.2.5 External policy agents

The MMS depends on administrative applications for some of the functionality of the overall system. These administrative applications are called External Policy Agents. Their purpose is to allow the MMS supplier or the local site to make decisions about how certain aspects of the MMS operate without requiring changes to the MMS implementation. Since external policy agents must use the MMP interface, they may be implemented in a variety of languages.

The following five external policy agents are described in 6.2.5.1 through 6.2.5.5.

- a) Cartridge life cycle
- b) Drive cleaning support
- c) Drive load balancing
- d) Task queue management
- e) Event aggregation and chargeback

MMS developers or local sites, as required, may define other external policy agents.

An MMS will function without external policy agents. However, without a cartridge life cycle policy agent, system operation will be cumbersome, so it is strongly recommended that this policy agent be present in all sites.

6.2.5.1 Cartridge life cycle

When the last VOLUME is deallocated from a cartridge, the cartridge is not made immediately available to be reused by another application. Some sites will want to perform special processing on the cartridge before letting it back into the set of unused cartridges.

An external policy agent shall query the MMS for all cartridges whose CARTRIDGE.*CartridgeState* field has the value *recycled*. This state will be set when the cartridge is owned by an application but does not have any VOLUMES associated with it. The external policy agent may perform any site-specific action that is necessary for those cartridges. Actions performed by the cartridge life cycle agent may range from

- a) Simply set the CARTRIDGE.*CartridgeState* field to *allocatable* and allow the cartridge to be reused.
- b) Overwrite the beginning (label portion) of the cartridge, and then set the CARTRIDGE.*CartridgeState* field to *allocatable* to allow the cartridge to be reused.
- c) Erase or degauss the entire cartridge, and then set the CARTRIDGE.*CartridgeState* field to *allocatable* to allow the cartridge to be reused.
- d) Eject the cartridge from the library, crush and burn the cartridge, and delete the CARTRIDGE from the MMS database.

If an external policy agent for cartridge life cycle management is not present, cartridges will not be reused without operator intervention.

6.2.5.2 Drive cleaning support

The MMS depends on external policy agents in order to support automated drive cleaning. The policy agent is free to examine any information inside or outside of the MMS and then to decide that a given drive needs cleaning. A site might choose a policy that all drives of a particular make/model will be cleaned every 60 hours of use while all other drives will be cleaned when the DM announces that the drive has decided it needs to be cleaned. The DM announces that fact by the addition of a *NEEDS_CLEANING* attribute on the DRIVECARTRIDGEACCESS record created by the most recent unmount operation.

The intent is that the policy agent will be the owner of all cleaning cartridge in the MMS, and it only needs to do a mount operation of one of those cartridges to effect a drive cleaning. Note that the policy agent may need to use the WHERE[] clause on the *mount* command.

6.2.5.3 Drive load balancing

The DRIVE.*DrivePriority* field in all DRIVE objects is intended to be periodically set by an external policy agent as a means of controlling the amount of usage that each drive is subject to in the MMS. The policy agent is free to examine any information inside or outside of the MMS and then to set the DRIVE.*DrivePriority* fields of any or all of the drives to new values.

When a mount request is being processed, each candidate drive in the system is examined and, among all drives that could be used, the MMS will choose to use the drive that has the lowest-valued *DrivePriority* field. For each set of drives that has the same *DrivePriority* value, drives will be used in an unspecified order. When each drive is first created, it is given a default *DrivePriority* value.

One site might choose to implement a drive usage policy that equalizes the amount of head wear across all equivalent drives, while another site might choose a policy that makes more intense use of the drives closer to the operator's station. If no drive load balancing policy agent is present, drive selection will occur, but the drive selection policy will be unspecified.

6.2.5.4 Task queue management

The *TASK.TaskPriority* field in all *TASK* objects may be modified by an external policy agent to control the order of execution of tasks that are queued. The policy agent is free to examine any information inside or outside of the MMS and then to set the *TASK.TaskPriority* fields of any or all of the tasks to new values.

When a resource becomes available, any pending tasks that can now be accommodated will be examined, and the MMS will select the task that has the lowest-valued *TaskPriority* field for execution. For tasks that have the same *TaskPriority* value, those tasks will be executed in first-come-first-served order. When each task is first queued, it is given a *TaskPriority* value based on the *AI.DefaultPriority* attribute.

One site might choose to implement a scheduling policy that gives preference to the task that has been waiting longest (this would be the case if no external policy agent was supplied for task queue management), while another site might choose a policy that lets the smallest tasks go first.

6.2.5.5 Event aggregation and chargeback

The MMS creates a *DRIVECARTRIDGEACCESS* object each time a cartridge is unmounted. Those objects contain information on drive and cartridge performance (transfer and error counts). An external policy agent is required to aggregate data on individual accesses to a cartridge or drives; aggregated statistics allow a site to notice and rectify potential wear and hardware problems. Also, the external policy agent may calculate and apportion usage costs based on site-specific formulae. The MMS makes no attempt to enforce any usage limits; it will only record the actual usage and allow that information to be accessed by an external policy agent.

6.2.6 Application start-up, shutdown, and abnormal termination

Applications are initiated by user actions or by system actions on the computer on which the application executes. When an application starts up, it obtains its network configuration, which specifies the network name or address of the MM, the registered name of the application and AI, and the secret key to be used when communicating with the MM. The method by which an application obtains its network configuration information is not specified by this standard; it might be in a per-application file, or stored in a system registry, for example. The application then contacts the MM using TCP/IP, utilizes the SSAIP and MMP to interact with the MM, says goodbye, and disconnects.

The following are three cases that shall be considered:

- a) The application terminates abnormally, or the system on which the application resides crashes.
- b) The MM terminates abnormally, or the system on which the application resides crashes.
- c) A network partition occurs, which makes it impossible for the application and MM to communicate with each other.

If the application terminates abnormally, the MM will detect this when the TCP connection is closed by the operating system of the application's computer. (The MM is constantly listening for activity, including a close operation, on all TCP connections to active applications, DMs, and LMs.) The MM cleans up any state that is related to the application and releases any resources (drives, cartridges) that are held by the application. The application should expect that any media that was mounted on its behalf would be unmounted by the MM. Note that the actions taken by the MM when an application terminates abnormally are identical to the actions it would take if the application unilaterally closed the TCP connection to the MM.

If the system on which the application is running crashes, the MM will detect this after some delay, and will then take appropriate action. The delay occurs because the MMS relies on the *keep-alive* function of the TCP protocol to detect host failure. The keep-alive mechanism indicates a disconnection when periodic keep-

alive packets are not acknowledged by the host operating system at the other end of the connection. Note that the duration of the delay between the system crash and the detection of the crash by the MM is determined by the keep-alive interval on the host on which the MM is running, and this may be a long interval. Typically, the keep-alive interval may be tuned on a per-system basis, and administrators for a computer that runs the MM may wish to consider shortening the keep-alive interval to a period of only a few minutes. Once a failed TCP connection is detected using the keep-alive mechanism, the MM will take the same actions as it does when an application crashes.

If the MM terminates abnormally, or if the machine on which the MM is running crashes, the application may continue running and using the drives and media it was assigned while the MM was running. The application may detect such a failure either by explicitly testing for a closed TCP connection (as the MM does), or it may simply notice a read or write failure when it attempts to communicate with the MM via the closed TCP connection.

It is safe for the application to continue using the devices it allocated while the MM is down as *long as these devices remain open by the application*. When the MM comes up after a crash, it utilizes stored information about its state prior to the crash. All logical mounts are placed into a *stale* category, and for each *stale* mount, the MM requests that the DM that is associated with the drive attempt to open the drive for exclusive use. If the DM is unable to do so, which would be the case if the application had the drive open, the MM repeats this request periodically. The request will eventually succeed when the application closes the drive.

A network partition is generally considered one of the more difficult cases to handle in a distributed system. Short-duration network partitions are not noticed by the MMS, as they are handled completely by the use of TCP. A network partition that exceeds the keep-alive interval will cause forcible disconnection of the MM and the application. The MM will behave in accordance with the rules given above, placing allocated devices into a *stale* state and attempting to re-establish communication with the DM associated with each stale device. The application will simply disconnect from the MM and go on with its processing.

6.2.7 Abnormal termination and restart of LMs and DMs

If an LM or a DM abnormally terminates, the application is not required to do anything. The MMS expects the LM or DM to be restarted by infrastructure on the host, and the restarted LM or DM will re-establish connection with the MM. The application will not be affected in any way. This process is explained in more detail in IEEE Std 1244.4-2000 and IEEE Std 1244.5-2000.

6.3 Media Manager (MM)

At the center of the MMS architecture is the *MM*. The MM is the central repository for metadata that describes the cartridges, applications, drives, libraries, and usage history of the MMS. It is also the access control manager for the resources assigned to the MMS, the arbitration engine for competing demands on those resources, and the coordinator that sequences the operations of all the LMs and DMs in the MMS on behalf of the MMP clients.

The MM shall have a persistent data store that can represent the information in the *MMS Data Model* as defined by this standard. The MM shall be able to simultaneously support the syntax and semantics of the following three interface protocols:

- a) *Media Manager Protocol* (IEEE Std 1244.3-2000)
- b) *Drive Manager Protocol* (IEEE Std 1244.4-2000)
- c) *Library Manager Protocol* (IEEE Std 1244.5-2000)

An MM that can support the syntax and semantics of the MMP but cannot support the DMP and LMP protocols may be said to partially conform to this standard. Such an MM would be a monolithic product with embedded LMs and DMs; it would only support applications that conform to the MMP (IEEE Std 1244.3-2000).

6.4 Library Managers (LMs) and Drive Managers (DMs)

The MMS makes use of one or more programs whose purpose is to manage a library or a drive on behalf of the MM. These programs are intended to be relatively small and simple to create.

An LM communicates with the MM via the LMP, as defined in IEEE Std 1244.5-2000. A DM communicates with the MM via the DMP, as defined in IEEE Std 1244.4-2000. Overviews of the LMP and DMP are provided in the paragraphs that follow.

The job of an LM is to represent the capabilities and contents of a library to the MM, and to implement the operations that the MM asks for, given the constraints of the library's structure. The job of a DM is to represent the capabilities of a drive to the MM, and to implement the operations that the MM asks for, given the constraints of the drive's capabilities.

The relationship between an LM or a DM and the MM is completely defined in their respective protocols. Both protocols define the relationship to be much more of a peer-to-peer style than a master-slave style. The LM or DM has the responsibility of implementing the MM's commands, but it can also initiate operations that the MM must fulfill.

The LMP defines an abstract model of a library and then defines operations on that model. The LM accepts those operations on the model and does whatever is needed to make the physical library do as close an approximation to that operation as is possible. A well-written LM would make the best operational use of all of the available features of the library, providing as much value as possible to the local systems' administration staff. The LMP also defines how the library and its contents are represented to the MM. The LM must generate a description, in terms defined by the LMP library model, of the cartridges, drives, and storage locations contained in the library.

The DMP defines an abstract model of a drive and then defines operations on that model. The DM accepts those operations on the model and does whatever is needed to make the physical drive perform as close an approximation to operations as possible. A well-written DM would make the best operational use of all of the available features of the drive, providing as much value as possible to the local systems administration staff. The DMP also defines how the drive and its capabilities or characteristics are represented to the MM. The DM must generate a description, in terms defined by the DMP drive model, of all the capabilities of the drive in order for the MM to know how this drive and its capabilities relate to the cartridges managed by the MM.

The internal structure of an LM or DM is not visible to the MM, nor is it relevant to the MMS as long as it abides by the protocol definition when it interacts with the MM. In order to accommodate the wide diversity of libraries and drives, a unique LM or DM may be created for each make and model of library or drive. Likewise, an LM or DM may be created to manage one of a family of similar devices. For example, a single LM could be written to handle both the 40 slot and the 80 slot models of a given company's automated library, while a different LM might be required to handle that same company's 240 slot automated library. As another example, a single DM could be written to handle both a DLT2000 and a DLT4000 drive, while a different DM might be required to handle a DLT7000 drive. Note that DLT drives are used here only as an example of drives belonging to a family.

6.4.1 Manager start-up, shutdown, and abnormal termination

In general, an MMS drive or LM process should be started whenever the system on which the process runs on boots up. Should a manager process terminate or be terminated, that process should be restarted as soon as possible.

Note that starting an MMS process does not imply that the process will become *active*. (Consider libraries and drives that may be accessed by multiple hosts.) The specific sequence of operations that describes what an LM or DM does during start-up is detailed in the standards for LMs and DMs.

6.4.2 Effect of abnormal termination and restart of system components on managers

If the MM abnormally terminates, each LM or DM is required to close its connection to the MMS and begin a new session.

7. Protocols

The interfaces between the major components of the MMS are defined as application-layer communication protocols, running over TCP/IP. The interfaces are session-oriented, and the TCP/IP connection is opened once and kept open throughout the session. The protocols are defined in a command-response manner, using character-based encoding similar to well-known Internet protocols, such as *telnet*, *ftp*, *smtp*, and *nntp*, rather than a binary-encoded RPC paradigm. MMS protocols are defined in terms of a formal, phrase-structured language, rather than a simple command structure, because the information being transferred across the MMS application-layer protocols is more complex than the information typically transferred using standard Internet protocols.

7.1 Language style

The three languages (MMP, LMP, and DMP) all have a common style; all three languages are text based and human readable, and there are common syntactic constructs. Note that there are no architectural requirements that the three protocols use the same style, but at present they are uniform in style. As the MMS evolves, one protocol might be changed to a different style—binary remote procedure call or XML-based, for example—while the others remain in the current style.

The only data type supported by these protocols is a string. In some cases the strings must be of a particular format as specified in 10.4. Using the MMP, an application can instruct the MM to make comparisons of strings based on a number of relational string comparison operators. Without a request for a relational comparison from an application, the only operation that the MM will perform on a string is a test of byte-for-byte equality/inequality with another string. Similarly, the only operations performed on strings within the LMP or DMP are tests of equality and inequality with another string. This common language style greatly simplifies the internationalization of string handling, as all code that performs multicharacter-set comparisons resides in the MM.

All *commands* in these protocols use a single token as the command name, have a set of *clauses* that follow the command, and a statement terminator. Typically, the clauses may be given in any order; in a few cases noted in the specification, clauses are executed sequentially and thus the order does make a difference. Each clause begins with a *reserved token* and, depending on the clause, may include subsidiary tokens and/or strings (for example, *arguments* to the clause).

This structure was chosen as a compromise between human readability and computer efficiency.

Here is an example, drawn from the LMP, where the reserved tokens and syntactic elements are shown in bold font and the string values are shown in *italics*:

```
move task[ "226" ] from[ "slot 12" "AB1234" ] to[ "slot 14" ] ;
```

In this example, the *move* command is followed by three *clauses*: *task*, *from*, and *to*. All strings are quoted and all unquoted tokens must be literally defined in the protocol definition.

7.2 Language descriptions: extended BNF

The IEEE MMS suite of standards use a modified version of Backus-Naur Form (BNF) to describe the syntax of languages. The meta-symbols of BNF⁸ are listed in Table 2.

Table 2—Meta-symbols of Backus-Naur Form (BNF)

::=	meaning “is defined as”
 	meaning “or”
<>	angle brackets used to surround category names

The angle brackets distinguish syntax rules names (also called nonterminal symbols) from terminal symbols, which are written exactly as they are to be represented. A BNF rule defining a nonterminal has the following form:

```
nonterminal ::= sequence_of_alternatives consisting of strings of
                terminals or nonterminals separated by the meta-symbol |
```

For example, the BNF production for a mini-language is as follows:

```
<program> ::=  program
                <declaration_sequence>
                begin
                <statements_sequence>
                end ;
```

This example shows that a mini-language program consists of the keyword *program* followed by the declaration sequence, the keyword *begin* and the statements sequence, and finally the keyword *end* and a semicolon. A traditional BNF definition of the syntax of the MMS protocols is not capable of conveniently representing the fact that, in some cases, clauses can occur in any order, but there can be only one occurrence of that clause in the command. Additional operators have been defined to allow these constraints to be expressed. The extended BNF used in these standards adds constructs to standard BNF shown in Table 3.

These notations may be combined and used with standard BNF syntax, for example:

⁸J. W. Backus, “The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM conference,” in *Proceedings of the International Conference on Information Processing (ICIP)*, Paris, June 1959.

Table 3—Extended BNF Symbols

Representation	Name	Description
[a]	Optional	Zero or one of a.
{ a b c }	Permutation	Represents the components a b and c in any order, e.g., a b c, a c b, b a c, b c a, c a b, or c b a. Note that this definition of <i>curly braces</i> differs from the definition of these symbols in other extended BNF versions.
(a b)	Grouping	Group the enclosed syntactic elements as a unit; typically used with the or operator “ ” and the repetition operators “*” and “+”. Note that the grouping operator is ordinary parentheses “()”. whereas the permutation operator is curly braces “{}”.
a*	Repetition	Zero or more of a.
a+	Repetition	One or more of a.

`<mycommand> ::= verb { <clause1> <clause2>* <clause3>+ }`

defines `<mycommand>` to consist of the keyword `verb` followed by, in any order, exactly one `<clause1>`, zero or more `<clause2>`, and one or more `<clause3>`.

`<anothercommand> ::= action [ <clauseX> | <clauseY> ]`

defines `<anothercommand>` as the keyword `action` followed by a clause that could be `<clauseX>`, `<clauseY>`, or nothing.

7.3 Message sequencing within protocols

The *MMP*, *DMP*, and *LMP* all use a three-phase message sequencing style. This sequencing style allows asynchronous execution of commands and receiver-controlled throttling of incoming command traffic. Each of the three protocols is bi-directional, so this sequencing applies to both directions simultaneously. The description below details how the sequencing works without reference to a particular protocol or a particular direction. Please refer to examples of message sequencing that are present in the examples of 4.4.1.

Commands are labeled with a *TaskID*. After each command is received at the other end, it is checked for syntax errors and an acknowledgment is sent back. Once the acknowledgment is received, another command can be sent. At some point in the future, possibly after many other commands have been sent, a response to the initial command will be received. (No response will occur if the original command was negatively acknowledged.) The response will be labeled with the unique TaskID that was used when the original command was sent. The key here is that multiple commands can be sent without waiting for a response, but an acknowledgment must be received for each command before the next one can be sent.

The fact that the acknowledgment of receipt is separate from the response to a command allows multiple commands to be outstanding at any instant. The receiver is not required to actually process those commands in parallel, but the protocol definition allows the receiver to do so if necessary.

The receiver of a command is free to delay sending an acknowledgment if necessary. This can be used to control the rate at which the other end of the connection is generating commands, as a new command may not be sent out until the acknowledgment of the prior command has been received. Restricting the sending of new commands in this way permits the implementer of a component to simplify the receiving code so that it

does not need to actually support the asynchronous execution of commands, or to control resource loading in the receiver.

In order to avoid a deadlock, each end of the communications channel must be prepared to accept input from the channel without regard to whether it is waiting on a response or an acknowledgment from the other end—TCP throttling must not be used to control generation of new commands. The socket input buffers cannot be allowed to fill up while the process is waiting to send data out the socket, and incoming commands must be processed even though the process is waiting on a response from the other end. Failure to heed this restriction could result in deadlock.

7.4 Security and authentication

Each computer system that is a part of the MMS has a protected file or its equivalent that contains keys used for authentication. The keys are tuples consisting of {host, client, instance, language}. These keys are used during the initial SSAIP authentication handshake.

The example in 4.4.1.1 shows how a session is initially established using SSAIP; further details on SSAIP are found in IEEE P1244.2/D041900.

7.5 Internationalization

The MMS includes features that allow relatively easy localization (internationalization) of user-visible messages. The following features are provided to readily permit localization:

- a) All communication between MMS components utilizes the ISO-10646 character set (Unicode) encoded in UTF-8 character representation. This allows Western character sets to be represented in a compact manner, yet permits the inclusion of all Unicode characters.
- b) Status messages are exchanged in a stylized format, including tokens that represent the meaning of the message in a unique, language-independent manner, and a vector of argument words that may be substituted in any order into a localized message for display to a human user.
- c) Centralized message catalogs are managed within the MM to allow different language localization to occur for different users of a single system.
- d) For messages that are a part of the IEEE MMS suite, English-language strings are defined for all messages. These strings are embedded in relevant parts of the IEEE MMS suite. Implementers are encouraged to use this English-language text in their implementations, but it is not required.
- e) Text in an implementation-defined default language may be propagated along with individual messages, so that in the event that a component of the system is upgraded and a language catalog has not been provided for that component in the desired language, a meaningful message can still be produced.

The defined tokens and syntactic elements in each protocol have been chosen from the US-ASCII subset, while the values contained in uninterpreted strings can be chosen from the full ISO-10646 character set (Unicode) encoded in UTF-8 character set.

The central message catalog, which is part of the MM, is the key to the localization of messages. Each system component, including LMs and DMs, may add new messages in one or more languages to the central message catalog. When a new LM or DM is installed, the corresponding message catalog should be installed in the central message catalog.

7.5.1 Language selection

The default language selection within the MM for a given installation is configurable. A client of the MM may request a change of language localization on a given connection using the *localize* command:

```
localize locale[ 'FR' ]
```

Multiple languages may also be requested:

```
localize locale[ 'FR' 'EN' ] sort[ 'UK' ]
```

If the MM has at least one message format stored in the requested language, an affirmative response is sent to the client. If the MM has no messages stored in the requested language, an error response is returned.

The two-character language identifier is selected using standard identifiers. See IETF RFC 1766-1995, ISO 639:1988, and ISO 3166:1993.

The *sort* clause of the *localize* command may also be used to select the sort order (character collating sequence) of certain operations:

```
localize sort[ 'FR' ]
```

Sort localization may not be supported by all MM implementations; a particular MM may or may not support sorting in a requested locale.

7.5.2 Localized messages

The MMS includes a comprehensive scheme that allows system components to be developed in a local language, and for messages from that component to be readily translated to and displayed in a language appropriate to the site of an installed system. Internationalization of localized messages is centralized in the MM, through the use of message catalogs. When a message passes through the MM from an LM or DM to an Application, the message catalog is consulted to produce a version of the message in the language(s) requested by the Application using the *locale* command.

A *message format string* describes each message, and the message format string is stored in multiple languages in the message catalog. The message format string consists of text plus a number of embedded tags; the tags refer to message arguments. The tags may be in different orders in different languages.

The following is a sample message in English:

```
The cartridge AB9278 in slot 12 bay 4 is stuck.
```

A message format string in the message catalog for the English version of this message might look as follows:

```
The cartridge $pcl$ in slot $slot$ bay $bay$ is stuck.
```

Substitution of words within the message format string is performed using a tag construct. Within the message format string, tags are defined using alphanumeric strings surrounded by dollar signs (e.g., \$pcl\$ or \$drive2\$). When a message format string is interpreted, these tags are removed and replaced with values of the corresponding name-value attribute pair from the *args* clause. If it is desired to include a dollar sign in the message format string, it must be escaped using a backslash character (\\$).

If the MM was communicating with a client that had requested German (DE) localization of messages, and the message catalog contained a message with the token *cartridge_stuck* in the language *DE*, the stored format might be

```
Es existiert ein Problem in Position $slot$/ $bay$. Die Bildplatte
$pcl$ in dieser Position konnte nicht entfernt werden.
```

Notice that the keys in the German message are in a different order than in the English message. When the MM produces the German message, it would be:

```
Es existiert ein Problem in Position 12/4. Die Bildplatte AB9278
in dieser Position konnte nicht entfernt werden.
```

Messages are represented using the following *message* clause:

```
message[ id[ 'stk' '9710' 'cartridge_stuck' ]
         arguments[ 'slot' '12' 'bay' '4' 'pcl' 'AB9278' ] ]
```

Note that message format strings exist *only* in the message catalog, which is managed by the MM—they are not transmitted as part of any language, nor are they visible to any component of the MMS other than the MM.

The manufacturer identifier (stk), catalog (9710), and message ID (cartridge_stuck) uniquely identify the message in the catalog as follows:

- a) The manufacturer identifier identifies the organization that defines the message. Manufacturer identifiers must be registered with the IEEE to ensure uniqueness. If the message is defined in the IEEE MMS suite of standards, the manufacturer identifier shall be *ieee*.
- b) The manufacturer defines the model identifier. A manufacturer may choose to have one or more message catalogs to aid in software maintenance and distribution. Taken together, the manufacturer identifier and model identifier are used to identify a message catalog. If the message is defined in the IEEE MMS suite of standards, the model shall be 1244.
- c) A code that uniquely identifies the message within the message catalog. Taken together, the manufacturer identifier, model, and code uniquely identify the exact error condition and the particular message associated with this condition.

If it is necessary for program logic to take action based on a specific message, care must be taken to ensure that non vendor-specific program logic rely only on IEEE 1244 defined message codes.

Message versioning may be accomplished by either never reusing message identifiers and extending an existing message catalog, or establishing a message catalog with a new catalog identifier (i.e., 9710, 9710a, 9710b, etc.).

The optional *arguments* clause consists of tag-value pairs; the tags in this example are *slot*, *bay*, and *pcl*.

Zero or more occurrences of the *loctext* clause may be present in a *message* clause. The *loctext* clause contains fully localized, clear-text versions of a message. The message originator (DM or LM) may supply a text clause in any desired language(s), or the LM or DM may choose to not supply a *loctext* clause. The MM will add and/or replace *loctext* clauses in the *message* with the language(s) requested by the application via the *locale* command to the message as it passes through the MM.

```
message[ id[ 'stk' '9710' 'cartridge_stuck' ]
arguments[ 'slot' '12' 'bay' '4' 'pcl' 'AB9278' ]
loctext[ 'EN'
        'The cartridge AB9278 in slot 12 bay 4 is stuck. ' ] ]
```

Most LMs and DMs should include a formatted message (*loctext*) in a default language; the *loctext* clause may be omitted for messages that originate on hardware devices with limited resources, such as drives or small libraries.

If the message originator generates only an *id* and *arguments*, then it is essential that the language catalog loaded into the MM include the specified message, or translation into a human-readable form will not be performed.

```
<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> "]"
```

7.5.3 Response messages

Response messages may be generated by any MMS component and passed to other MMS components. These messages may or may not be intended for eventual human use.

Response messages may occur

- a) *To reject command input*—This is typically because the input string cannot be parsed, or because input was received before acknowledgment of a prior command; this is the **unacceptable** response message.
- b) *To accept a command for processing*—This is the **accepted** response message.
- c) *To transmit the result of a command*—This may be **success** in the case of a successful command completion, **cancelled** if a command was cancelled using the **cancel** command, or **error** if the command terminated abnormally.

The **response** command is defined as follows:

```
<response-message> ::= "response" { <negative-response> |
                                <accepted-response> |
                                <command-response> } ";"
<negative-response> ::= { "unacceptable" [ <message-clause> ] }
<accepted-response> ::= { <task-clause> "accepted" }
<command-response> ::= { <task-identifier>
                        <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> "]"

```

Class and *code* are tokens that are defined in the MMP, LMP, and DMP. These tokens are intended for machine processing of a response message in a standardized manner.

A *message-clause* should be present in any response message that might be intended for eventual human consumption.

7.5.4 Command sequencing

The normal sequence of events when system component A sends a command to system component B is as follows:

- a) A sends the command to B.
- b) If the command is syntactically incorrect or must be rejected immediately, B sends an **unacceptable** response to A. A is then free to send further commands to B.
- c) If the command is syntactically correct, B sends an **accepted** response to A. A is then free to send further commands to B. If the command is syntactically correct, but before accepting the command, B must abort processing of the command, B sends a **cancelled** response to A.
- d) If the command completes successfully, B sends a **success** reply to A. If the command fails to complete, B sends an error reply to A.

7.6 Overview of SSAIP

The IEEE SSAIP is used by the IEEE MM when an MMS Client or an MMS Module wishes to connect to the MM.

The SSAIP provides identification, and if desired, authentication of the client, which is a requirement to obtain access to the services of the MM in compliance with the MMS security model. The SSAIP also establishes parameters of the communications between the MMS Client and the MMS Module thereafter, such as language and language type.

An example of operation of the SSAIP is given in 4.4.1.

IEEE P1244.2/D041900 defines SSAIP.

7.7 Overview of MMP

The IEEE MMS MMP is the language that MMS client and administrative programs use to communicate with the MM.

The MM manages media and provides access to media when requested. The MMP language provides capabilities in the following areas:

- a) *Media Management* functions for principals (applications) of the MMS to access and manage removable media.
- b) *Task Management* functions for applications to control the system and the environment in which they operate.
- c) *Operator Interaction* functions for operations personnel to perform routine tasks.
- d) *Device Management* functions for the control of devices.
- e) *System Management* functions for tuning the operation of the MM.
- f) *Data Management* functions for implementing administrative-level capabilities, and for applications to view and modify meta-data associated with the media.

The IEEE Std 1244.3-2000 defines the MMP. At the time this standard was published, the MMP included the following commands; an asterisk (*) indicates that a command is privileged:

- *Media Management*: begin, end, mount, unmount, allocate, deallocate, rename
- *Session Management*: hello, privilege, locale, goodbye
- *Task Management*: cancel, shutdown*
- *Data Management*: attribute, show, create*, delete*
- *Device Management*: cpattribute*, cpshow*, cpreset*, cpscan*
- *Operator Interface*: accept*, respond*, release*, move*, eject*, inject*

An example of operation of the MMP is given in 4.4.1.

7.8 Overview of DMP

The DMP is the language that the MM uses to communicate with DMs. The DMP defines the abstraction that the MMS utilizes for drive configuration and control. This common drive interface is used for all drives in the MMS. Interface uniformity at the drive level allows for the simple addition of new managed drives without the necessity of re-compilation of any modules in the MM.

DMP consists of two languages: DMP/M, which includes commands from the MM to the DM, and DMP/D, which includes commands from the DM to the MM.

IEEE Std 1244.4-2000 defines the DMP. At the time this standard was published, the DMP included the following commands:

- a) *DMP/M*: attach, detach, load, unload, activate, reset, exit, private, cancel
- b) *DMP/D*: config, message, ready, private

A brief example of operation of the DMP is given in 4.4.1.

7.9 Overview of LMP

The LMP is the language that the MM uses to communicate with LMs. The LMP defines the abstraction that the MMS utilizes for Library configuration and control. This common Library interface is used for all Libraries in the MMS. Interface uniformity at the Library level allows for the simple addition of new managed Libraries without the necessity of re-compilation of any modules in the MM.

LMP consists of two languages: LMP/M, which includes commands from the MM to the LM, and LMP/L, which includes commands from the LM to the MM.

IEEE Std 1244.5-2000 defines the LMP. At the time this standard was published, the LMP included the following commands:

- a) *LMP/M*: mount, unmount, move, openport, scan, activate, reset, exit, barrier, private, cancel
- b) *LMP/L*: message, config, ready, private, cancel

A brief example of operation of the LMP is given in 4.4.1.

7.10 Overview of MMIP

The Media Manager Interchange Protocol (MMIP) defines a scheme for moving all metadata associated with one or more volumes and cartridges from one MM to another. The metadata is extracted in a portable manner and may be sent by e-mail or by using magnetic media, such as a floppy disk, which can be sent along with the physical media.

The MMIP allows critical life cycle information, such as usage, wear, and error data, to remain with media when it is moved from the control of one MMS to another.

IEEE P1244.6 will define the MMIP.

7.11 Overview of MMCIP

The Media Manager Control Interface Protocol (MMCIP) defines a mechanism that allows an external MM, such as media management services running on an IBM mainframe, to be used to manage media in conjunction with an IEEE MMS. The MMCIP consists of a set of back-end database and control operations that the MMS will use to store common metadata information and to perform physical operations on cartridges that are managed by the external MM.

IEEE P1244.7 will define the MMCIP.

8. Tokens

The current, official registry of tokens is maintained on the Web site <http://www.ieee-sssc.org>, which is operated by the IEEE Storage System Standards Committee (SSSC). A list of tokens is given in Clause 20, which represents the tokens in existence at the time of publication of the standard.

If a new token is needed, it can be either unregistered or registered. Unregistered tokens must use a prefix (e.g., SGI_, SUN_, etc.) that is registered. To register a token or a prefix, see the instructions on the Web site. It is strongly encouraged that a new token be created only if the existing tokens, or a grouping of them, do not adequately cover the desired semantic.

9. Data model—introduction

The data model defines the inner workings of the MMS, including what essential information is store, and how different components, both physical and logical, relate to one another. This standard defines the required components of the data model, but it does not require a particular implementation of the data model. Possible choices for a particular implementation of the MMS might include an in-memory database,

flat files, and a relational database. Individual implementations may also need to add attributes to the data model; however, such additional attributes shall not affect the standardized functioning of the interfaces. Extensions to interface languages, which are denoted by languages with names different from the standard IEEE language names, may require extensions to the data model. Any such extensions shall not affect the standardized functioning of standard IEEE languages that are supported by an implementation.

This standard specifies both an abstract model, as well as those parts of a concrete implementation of that model that are thought necessary to be standardized so that different components of the MMS may communicate with each other. In order for implementations of different components of the MMS from different vendors to communicate with each other, there must be agreement on the objects, the attributes, their names, the data types, and the semantics as described in this standard. The data types specified for the attributes of the objects constrain only the external representation of those attributes. The internal representation of the attributes is not specified by this standard.

The data model consists of a number of objects, each object having a collection of attributes. This scheme correlates well with the relational model, where each table has a number of columns and rows. Each row contains one data object—a tuple that describes a number of attributes that are part of that object. An object may have both required and optional attributes. The MMS data model extends the standard relational model by allowing, for some tables, the addition of new attributes to a given object. These user-defined attributes are one of the primary extension mechanisms within the MMS. User-defined attributes need not be present for all rows in a table.

For example, one particular model of tape drive may wish to store a value called “magnetic_flux_field” for each cartridge that it has written. The manufacturer of this product, the Acme tape company, could define a column as “_acme_magnetic_flux_field”, and then place a value for this column in the tape cartridges that were written by Acme’s tape drive. This column, however, would not be present for non-Acme cartridges.

9.1 Clusters and objects

The 40 different types of objects that an MMS supports are subdivided into 8 different clusters of related objects. The clusters closely match the intuitively distinct pieces of an MMS. The clusters are summarized in Table 4.

Table 4—Clusters of objects an MMS supports

Cluster name	Object names	Types of objects in that cluster
Application cluster	APPLICATION, AI	Related to the definition and authorization of client programs.
Library cluster	LIBRARY, LM, BAY, SLOT, SLOT-GROUP, SLOTCONFIG, SLOTTYPE	Related to libraries, generally defined and updated by an LM.
Drive cluster	DRIVE, DRIVEGROUP, DRIVEGROUPAPPLICATION, DM, DMBITFORMAT, DMBITFORMATTOKEN, DMCAPABILITY, DMCAPABILITYTOKEN, DMCAPABILITYDEFAULTTOKEN, DMCAPABILITYGROUP, DMCAPABILITYGROUPTOKEN	Related to drives, generally defined and updated by a DM.

Table 4—Clusters of objects an MMS supports (continued)

Cluster name	Object names	Types of objects in that cluster
Cartridge cluster	CARTRIDGE, CARTRIDGEGROUP, CARTRIDGEGROUPAPPLICATION, SIDE, PARTITION, VOLUME, CARTRIDGETYPE	Related to cartridges and what they contain.
Session cluster	CONNECTION, SESSION	Related to currently connected and/or active clients.
Mount cluster	MOUNTPHYSICAL, MOUNTLOGICAL, DRIVECARTRIDGEACCESS	Related to currently mounted volumes and cartridges.
Task cluster	TASK, TASKCARTRIDGE, TASKDRIVE, TASKLIBRARY	Related to commands currently blocked and waiting for resources.
System cluster	MESSAGE, REQUEST, SYSTEM, STALEHANDLE	Related to system configuration.

9.2 Relationships between clusters

The relationship between any two objects in any two clusters is defined to be the same as any other two objects in those same clusters. Another way to say it is that the clusters have relationships to each other that are not dependent on any specified objects in those two clusters.

When more than two clusters are discussed at the same time, the overall relationships are simply the sum of the relationships between each of the distinct pairs of clusters. Each successive cluster being considered effectively adds more layers of filtering (exclusion) on top of the restrictions that were there before.

Table 5 reveals the relationships of clusters of objects. Cluster names are shown in bold on a left-to-right descending diagonal (e.g., “Application”). The relationships can best be expressed as restrictions on the possible combinations of objects. Given two objects, if the specified condition is not true, then those two objects are unrelated.

Table 5—Relationships of clusters of objects

Application	Application may use a cartridge or drive in the library.	Application may use the drive.	Application has a volume in or may allocate a volume in the cartridge.	Application is being used in the session.	Application caused the mount.	Application submitted the command that started the task.
	Library	Library contains the drive.	Library contains the cartridge.	Library is being used by the session.	Mount occurred on a drive in the library.	Library is involved in executing the task.
		Drive	Cartridge is in the drive.	Drive is being used in the session.	Drive is being used by this mount.	Drive is involved in executing the task.
	Cartridge		Cartridge is being used in the session.	Cartridge is being used by this mount.	Cartridge is involved in executing the task.	
		Session	Session has caused this mount.	Session has caused this task to be performed.		
	Mount		The task involves this mount.			
		Task				
Configuration						