Working Draft

Project T10/1355-D

Revision 06
23 August 2002

Information technology - SCSI Object-Based Storage Device Commands (OSD)

T10 Technical Editor:

Ralph O. Weber
ENDL Texas
18484 Preston Road
Suite 102 PMB 178
Dallas, TX 75252
USA

Telephone: 214-912-1373
Facsimile: 972-596-2775
Email: ROWeber@ACM.org

Funded by Intel Labs

Reference number
ISO/IEC 14776-313 : 200x
ANSI INCITS.*:200x**

Printed Friday, August 23, 2002 11:19 AM

Points of Contact:

T10 Chair
John B. Lohmeyer
LSI Logic
4420 Arrows West Drive
Colorado Springs, CO 80907-3444
Tel: (719) 533-7560
Fax: (719) 533-7183
Email: lohmeyer@t10.org

T10 Vice-Chair
George O. Penokie
IBM
3605 Highway 52 N
MS: 2C6
Rochester, MN 55901
Tel: (507) 253-5208
Fax: (507) 253-2880
Email: gop@us.ibm.com

INCITS **Secretariat**
INCITS Secretariat
1250 Eye Street, NW Suite 200
Washington, DC 20005
Telephone: 202-737-8888
Facsimile: 202-638-4922
Email: incits@itic.org

T10 Web Site www.t10.org

T10 Reflector To subscribe send e-mail to majordomo@T10.org with 'subscribe' in message body
To unsubscribe send e-mail to majordomo@T10.org with 'unsubscribe' in message body
Internet address for distribution via T10 reflector: T10@T10.org

Document Distribution

INCITS Online Store
managed by Techstreet
1327 Jones Drive
Ann Arbor, MI 48105
http://www.techstreet.com/incits.html
Telephone: 1-734-302-7801 or
1-800-699-9277
Facsimile: 1-734-302-7811

or

Global Engineering
15 Inverness Way East
Englewood, CO 80112-5704
http://global.ihs.com/
Telephone: 1-303-792-2181 or
1-800-854-7179
Facsimile: 1-303-792-2192

Additional Work

This working draft is an incomplete work in progress.

Areas known to require additional work are:

a) Sessions
b) Capabilities
c) Policies & interfaces to the policy manager
d) Security
e) Placement attributes
f) Quota definition and enforcement
g) Should OSD use the SBC options bits?
h) Should data transfer commands include a priority field in the CDB?
i) Model for bi-directional and segmented buffers

Completion of the outstanding work in any of these areas may result in changes to CDB and parameter data formats.

Revision Information

1 Approved Documents Included

The following T10 approved proposals have been incorporated SPC-3 up to and including this revision:

02-130r0 SBC-2, SSC-2, SMC-2 & OSD Support for All Registrants Persistent Reservations

To the best of the technical editor's knowledge, the following T10 proposals have been approved for inclusion in SPC-3 but not included in this revision:

none

2 Revision History

2.1 Revision 0 (Gene Milligan)

Initial draft

2.2 Revision 1 (Gene Milligan)

Converted to ISO/IEC style.
Edits agreed in 1/2000 T10 meeting.

2.3 Revision 2 (Gene Milligan)

General edits through 5.2.
Item 1-d of 00-262r0 per 7/2000 T10 OSD WG.
Item 1-e of 00-262r0 per 7/2000 T10 OSD WG.
Item 2.1 and 2.3 of 00-262r0 per 7/2000 T10 OSD WG.
Item 3.4 of 00-262r0 per 7/2000 T10 OSD WG.
Added reservations clause as a point of departure.

Modified abstract partially based upon comments from John Wilkes of HP and made other edits as suggested by some of his comments.

Eliminating footnotes.

2.4 Revision 3 (Gene Milligan)

Markups from 9/14/2000 T10 working group.
T10/00-330r0
General edits from 5.3 to the end of the draft.
Added acronyms per working group request.
Added hierarchy diagram to conventions and model per working group request.

2.5 Revision 4 (4 July 2001)

Editorship assumed and revision 3 converted to FrameMaker [Ralph Weber].
Front matter through Clause 3 generally adapted from SPC-2 revision 19 [Ralph Weber].
Merge Annex C section 6 (Security) into main draft for community consideration [Garth Gibson].
Numerous editorial corrections and editor's notes added [Ralph Weber].

2.6 Revision 5 (29 March 2002)

The names of several OSD objects changes as follows:

OSD r04	Proposed on OSD Reflector	Acceptable to T10 LB reviewers	After Further Consideration
object	no acronym needed		
root object	RootObject	Root_Object	root object
group object	GroupObject	Group_Object	group object
user object	UserObject	User_Object	user object
user object id	UserObjectID	User_Object_ID	User_Object_ID
(as object ids only apply to user objects)			
object group	ObjectGroup (instead of OG)	Object_Group	object group
object group id	ObjectGroupID	Object_Group_ID	Object_Group_ID
object session	ObjectSession	Object_Session	object session
(session is an iSCSI term)			
object session id	ObjectSessionID	Object_Session_ID	Object_Session_ID

The following changes have been made with respect to the response data:

a) Remove SCSI Status from all Response Data format tables;
b) If removing SCSI Status reduces a Response Data format table to null remove the table;
c) If removing SCSI Status does not reduce the Response Data format table to null add a RESPONSE ALLOCATION LENGTH field to the CDB definition and reference SPC-3 for its usage; and
d) Remove Response Data from the READ command (no exceptions).

A “Parameter Data” clause was added to describe the diagnostic page, log page, and mode page formats supported by OSD devices. The clause contents were copied with virtually no modifications from a similar SPC-3 clause for processor type devices.

All text with strike throughs in r04 has been removed.

Acted to resolve as many editors notes in the model section as possible.

I the Command Formats and Command sections, the CDB structure was made completely consistent across all OSD service actions. Likewise, the response data format was made completely consistent wherever used. The capabilities parameters were defined in detail. The Commands section was alphabetized. Descriptions were added for all fields.

The structure of an OSD time value was defined.

The specifics of digital signatures were nailed down, including normative references to HMAC-SHA1 and 3DES.

A PRIORITY field was added to the READ, WRITE, and APPEND service actions, as hinted at in the description of the READ command.

The LIST service action was given a CDB format.

The FORMAT OSD service action was given response data.

A CREATE AND WRITE service action was added so that the ATTRIBUTES MASK field could be dropped from its CDB format. One command cannot both write user data and include a parameter list and the presence of an ATTRIBUTES MASK field implies the need for a parameter list (see the SET ATTRIBUTES service action).

In numerous places, 'object' was changed to 'user object' because it appeared that the usage of 'object' was not intended to include the root object or any specific group objects.

2.7 Revision 6 (12 July 2002)

Revision 6 incorporates the following T10 approved proposals:

02-130r0 SBC-2, SSC-2, SMC-2 & OSD Support for All Registrants Persistent Reservations

The major effort in revision 6 is the definition of attribute pages and attribute lists. Also, all commands have been given the ability to get and set attributes, with parameter and data transfer commands having a slightly more restricted ability than commands that do not transfer parameters or data. The necessary changes appear in the model, commands, and parameter data clauses with the bulk of the changes in the parameter data clauses.

Revision 6 contains an agreed root, group, and user object addressing mechanism. As part of the agreement, the concept of a default object group has been removed.

ANSI (r)
INCITS.*:200x**

Draft

American National Standards
for Information Systems -

SCSI Object-Based Storage Device Commands (OSD)

Secretariat
National Committee for Information Technology Standards

Approved mm dd yy

American National Standards Institute, Inc.

Abstract

This SCSI command set is designed to provide efficient peer-to-peer operation of input/output logical units that manage the allocation, placement, and accessing of variable-size data-storage containers, called objects. Objects are intended to contain operating system and application constructs.

Draft

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer. Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered and that effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he or she has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not confirming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give interpretation on any American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken periodically to reaffirm, revise, or withdraw this standard. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

CAUTION: The developers of this standard have requested that holders of patents that may be required for the implementation of the standard, disclose such patents to the publisher. However, neither the developers nor the publisher have undertaken a patent search in order to identify which, if any, patents may apply to this standard.

As of the date of publication of this standard and following calls for the identification of patents that may be required for the implementation of the standard, no such claims have been made. No further patent search is conducted by the developer or the publisher in respect to any standard it processes. No representation is made or implied that licenses are not required to avoid infringement in the use of this standard.

Published by
American National Standards Institute
11 West 42nd Street, New York, NY 10036

Printed in the United States of America

Contents

Tables

Figures

Foreword

This foreword is not part of American National Standard INCITS.***:200x.

This SCSI command set is designed to provide efficient peer-to-peer operation of input/output logical units that manage the allocation, placement, and accessing of variable-size data-storage containers, called objects. Objects are intended to contain operating system and application constructs.

This SCSI command set provides multiple operating systems concurrent control over one or more input/output logical units. However, the multiple operating systems are assumed to properly coordinate their actions to prevent data corruption. This SCSI standard provides commands that assist with coordination between multiple operating systems. However, details of the coordination are beyond the scope of the SCSI command set.

This standard defines a logical unit model for Object-Based Storage Device logical units. Also defined are SCSI commands that apply to Object-Based Storage Device logical units.

Objects designate entities in which computer systems store data. The purpose of this abstraction is to assign to the storage device the responsibility for managing where data is located on the device.

This standard was developed by T10 in cooperation with industry groups during 1999 through 200X. Most of its features have been tested in pilot products implementing these concepts in conjunction with standard transport protocols.

With any technical document there may arise questions of interpretation as new products are implemented. INCITS has established procedures to issue technical opinions concerning the standards developed by INCITS. These procedures may result in SCSI Technical Information Bulletins being published by INCITS.

These Bulletins, while reflecting the opinion of the Technical Committee that developed the standard, are intended solely as supplementary information to other users of the standard. This standard, ANSI INCITS.***:200x, as approved through the publication and voting procedures of the American National Standards Institute, is not altered by these bulletins. Any subsequent revision to this standard may or may not reflect the contents of these Technical Information Bulletins.

Current INCITS practice is to make Technical Information Bulletins available through:

INCITS Online Store	http://www.techstreet.com/incits.html
managed by Techstreet	Telephone: 1-734-302-7801 or
1327 Jones Drive	1-800-699-9277
Ann Arbor, MI 48105	Facsimile: 1-734-302-7811

or

Global Engineering	http://global.ihs.com/
15 Inverness Way East	Telephone: 1-303-792-2181 or
Englewood, CO 80112-5704	1-800-854-7179
	Facsimile: 1-303-792-2192

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent to the INCITS Secretariat, National Committee for Information Technology Standards, Information Technology Institute, 1250 Eye Street, NW, Suite 200, Washington, DC 20005-3922.

This standard was processed and approved for submittal to ANSI by the InterNational Committee for Information Technology Standards (INCITS). Committee approval of the standard does not necessarily imply that all committee members voted for approval. At the time of it approved this standard, INCITS had the following members:

<<Insert INCITS member list>>

Technical Committee T10 on Lower Level Interfaces, which developed and reviewed this standard, had the following members:

John B. Lohmeyer, Chair
George O. Penokie, Vice-Chair
Ralph O. Weber, Secretary

<<Insert T10 member list>>

Introduction

The SCSI Object-Based Storage Device Commands (OSD) standard is divided into six clauses:

Clause 1 is the scope.
Clause 2 enumerates the normative references that apply to this standard.
Clause 3 describes the definitions, symbols, and abbreviations used in this standard.
Clause 4 describes the model for an OSD device and the conceptual relationship between this document and the SCSI-3 Architecture Model.
Clause 5 describes the data formats used throughout this standard.
Clause 6 describes commands that may be implemented by an OSD SCSI device.
Clause 7 defines the parameter data formats that may be implemented by an OSD SCSI device.

The annexes provide information to assist with implementation of this standard.

American National Standard **INCITS.***:200x**

American National Standard for Information Systems - Information Technology - SCSI Object-Based Storage Device Commands (OSD)

1 Scope

This standard defines the command set extensions to control operation of Object-Based Storage devices. The clause(s) of this standard pertaining to the SCSI Object-Based Storage Device class, implemented in conjunction with the applicable clauses of the ISO/IEC 14776-312 SCSI Primary Commands -2 (SPC-2), fully specify the standard command set for SCSI Object-Based Storage devices.

The objective of this standard is to provide the following:

a) Permit an application client to communicate with a logical unit that declares itself to be a Object-Based Storage device in the device type field of the INQUIRY command response data over an SCSI service delivery subsystem;
b) Enable construction of a shared storage processor cluster with equipment and software from many different vendors;
c) Define commands unique to the type of SCSI Object-Based Storage devices;
d) Define commands to manage the operation of SCSI Object-Based Storage devices.

The set of SCSI standards specifies the interfaces, functions, and operations necessary to ensure interoperability between conforming SCSI implementations. This standard is a functional description. Conforming implementations may employ any design technique that does not violate interoperability.

Figure 1 shows the relationship of this standard to the other standards and related projects in the SCSI family of standards as of the publication of this standard.

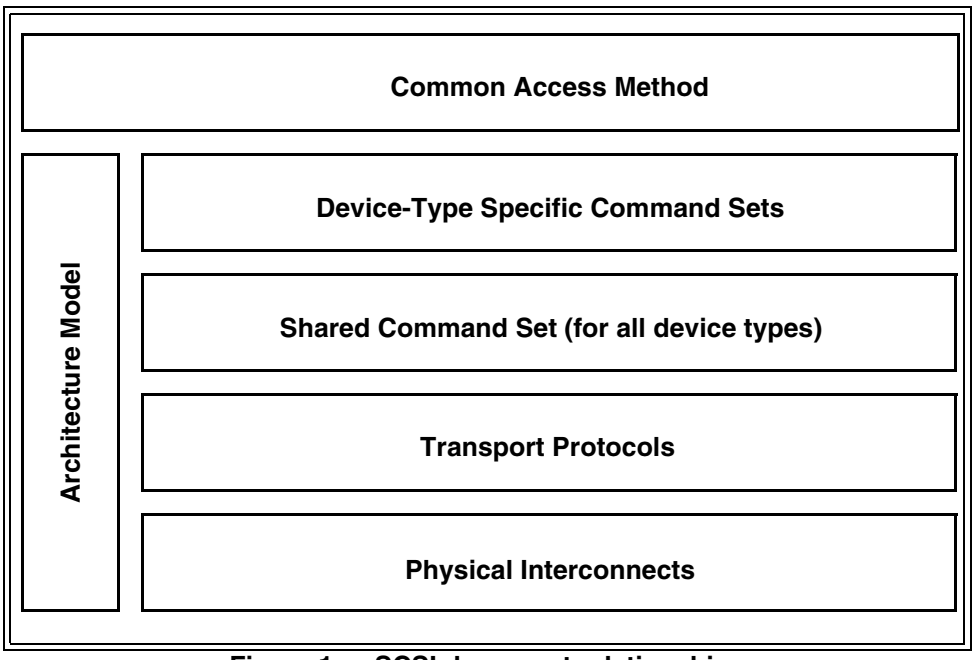

Figure 1 — SCSI document relationships

Figure 1 is intended to show the general relationship of the documents to one another. Figure 1 is not intended to imply a relationship such as a hierarchy, protocol stack, or system architecture. It indicates the applicability of a standard to the implementation of a given transport.

At the time this standard was generated, examples of the SCSI general structure included:

Interconnects:		
Automation/Drive Interface - Physical Layer	ADP	[T10/1556-D]
Fibre Channel Arbitrated Loop	FC-AL	[ANSI X3.272-1996]
Fibre Channel Arbitrated Loop -2	FC-AL-2	[ISO/IEC 14165-122] [ANSI NCITS.332-1999]
Fibre Channel Physical and Signalling Interface	FC-PH	[ISO/IEC 14165-111] [ANSI X3.230-1994]
Fibre Channel Physical Amendment 1		[ANSI X3.230/AM1-1996]
Fibre Channel 3rd Generation Physical Interface	FC-PH-3	[ISO/IEC 14165-113] [ANSI X3.303-1998]
Fibre Channel Physical Interfaces	FC-PI	[T11/1235-D]
Fibre Channel Framing and Signaling Interface	FC-FS	[T11/1331-D]
High Performance Serial Bus		[ANSI IEEE 1394-1995]
High Performance Serial Bus (supplement to ANSI/IEEE 1394-1995)		[ANSI IEEE 1394a-2000]
SCSI Parallel Interface - 2	SPI-2	[ISO/IEC 14776-112] [ANSI X3.302-1999]
SCSI Parallel Interface - 3	SPI-3	[ISO/IEC 14776-113] [ANSI NCITS.336-2000]
SCSI Parallel Interface - 4	SPI-4	[ISO/IEC 14776-114] [ANSI INCITS.362-200x]
SCSI Parallel Interface - 5	SPI-5	[ISO/IEC 14776-115] [T10/1525-D]
Serial Storage Architecture Physical Layer 1	SSA-PH	[ANSI X3.293-1996]
Serial Storage Architecture Physical Layer 2	SSA-PH-2	[ANSI NCITS.307-1998]
Serial Attached SCSI	SAS	[T10/1562-D]
SCSI Protocols:		
Automation/Drive Interface - Transport Protocol	ADT	[T10/1557-D]
Serial Storage Architecture Transport Layer 1	SSA-TL-1	[ANSI X3.295-1996]
Serial Storage Architecture Transport Layer 2	SSA-TL-2	[ANSI NCITS.308-1998]
SCSI-3 Fibre Channel Protocol	FCP	[ISO/IEC 14776-221] [ANSI X3.269-1996]
SCSI Fibre Channel Protocol - 2	FCP-2	[ISO/IEC 14776-222] [ANSI NCITS.350-200x]
SCSI Fibre Channel Protocol - 3	FCP-3	[ISO/IEC 14776-223] [T10/1560-D]
Serial Bus Protocol - 2	SBP-2	[ISO/IEC 14776-232] [ANSI NCITS.325-1999]
Serial Bus Protocol - 3	SBP-3	[ISO/IEC 14776-233] [T10/1467-D]
Serial Storage Architecture SCSI-3 Protocol	SSA-S3P	[ANSI NCITS.309-1998]
SCSI on Scheduled Transfer	SST	[T10/1380-D]
SCSI RDMA Protocol	SRP	[T10/1415-D]
Shared Command Sets:		
SCSI-3 Primary Commands	SPC	[ISO/IEC 14776-311] [ANSI X3.301-1997]

SCSI Primary Commands - 2	SPC-2	[ISO/IEC 14776-312] [ANSI NCITS.351-2001]
SCSI Primary Commands - 3	SPC-3	[ISO/IEC 14776-313] [T10/1416-D]

Device-Type Specific Command Sets:

SCSI-3 Block Commands	SBC	[ISO/IEC 14776-321] [ANSI NCITS.306-1998]
SCSI Block Commands - 2	SBC-2	[ISO/IEC 14776-322] [T10/1417-D]
SCSI-3 Stream Commands	SSC	[ISO/IEC 14776-331] [ANSI NCITS.335-2000]
SCSI Stream Commands - 2	SSC-2	[ISO/IEC 14776-332] [T10/1434-D]
SCSI-3 Medium Changer Commands	SMC	[ISO/IEC 14776-351] [ANSI NCITS.314-1998]
SCSI Media Changer Commands - 2	SMC-2	[ISO/IEC 14776-352] [T10/1383-D]
SCSI-3 Multimedia Command Set	MMC	[ANSI X3.304-1997]
SCSI Multimedia Command Set - 2	MMC-2	[ISO/IEC 14776-362] [ANSI NCITS.333-2000]
SCSI Multimedia Command Set - 3	MMC-3	[ISO/IEC 14776-363] [T10/1363-D]
SCSI-3 Controller Commands	SCC	[ISO/IEC 14776-341] [ANSI X3.276-1997]
SCSI Controller Commands - 2	SCC-2	[ISO/IEC 14776-342] [ANSI NCITS.318-1998]
SCSI Reduced Block Commands	RBC	[ISO/IEC 14776-326] [ANSI NCITS.330-2000]
SCSI-3 Enclosure Services Commands	SES	[ISO/IEC 14776-371] [ANSI NCITS.305-1998]
SCSI Enclosure Services Commands -2	SES-2	[ISO/IEC 14776-372] [T10/1559-D]
SCSI Specification for Optical Card Reader/Writer	OCRW	[ISO/IEC 14776-381]
Object-based Storage Devices Commands	OSD	[T10/1355-D]
SCSI Management Server Commands	MSC	[T10/1528-D]
Automation/Drive Interface - Commands	ADC	[T10/1558-D]

Architecture Model:

SCSI-3 Architecture Model	SAM	[ISO/IEC 14776-411] [ANSI X3.270-1996]
SCSI Architecture Model - 2	SAM-2	[ISO/IEC 14776-412] [T10/1157-D]
SCSI Architecture Model - 3	SAM-3	[ISO/IEC 14776-413] [T10/1561-D]

The term SCSI is used to refer to the family of standards described in this clause. The Small Computer System Interface - 2 standard (ANSI X3.131-1994) and the architecture that it describes are referred to herein as SCSI-2.

2 Normative references

2.1 Normative references

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

Copies of the following documents may be obtained from ANSI: approved ANSI standards, approved and draft international and regional standards (ISO, IEC, CEN/CENELEC, ITUT), and approved and draft foreign standards (including BSI, JIS, and DIN). For further information, contact ANSI Customer Service Department at 212-642-4900 (phone), 212-302-1286 (fax) or via the World Wide Web at http://www.ansi.org.

2.2 Approved references

ISO/IEC 60027-2-am2 (1999-01), Letter symbols to be used in electrical technology - Part 2: Telecommunications and electronics (Amendment 2)

ISO/IEC 60027-2-am2 (1999-01), Letter symbols to be used in electrical technology - Part 2: Telecommunications and electronics (Amendment 2)

ISO/IEC 14776-312, SCSI Primary Commands -2, (SPC-2) [ANSI NCITS.351:200x]

ANSI X9.52-1998, Triple Data Encryption Algorithm Modes of Operation

FIPS 180-1 (1995), Secure Hash Standard (HMAC-SHA1)

2.3 References under development

At the time of publication, the following referenced standards were still under development. For information on the current status of the document, or regarding availability, contact the relevant standards body or other organization as indicated.

ISO/IEC 14776-412, SCSI Architecture Model - 2 (SAM-2) [T10/1157-D]

3 Definitions, symbols, abbreviations, and conventions

3.1 Definitions

3.1.1 additional sense code: A combination of the additional sense code and additional sense code qualifier fields in the sense data (see 3.1.38).

3.1.2 application client: An object that is the source of SCSI commands. Further definition of an application client may be found in SAM-2.

NOTE 1 In typical networking applications a network client putting its workload on a server, which in turn submits I/O requests to storage, is not the applicable application client. The network client's server is, as the server is the element actually engaging in I/O with the OSD.

3.1.3 attributes: Data that is association with an OSD object (i.e., root, group, or user) that is not accessible directly via READ and WRITE commands, sometimes called metadata. See 4.4.3.

3.1.4 autosense data: The sense data that is automatically delivered to the application client by the device server in a protocol specific manner when a command completes with a CHECK CONDITION status (see SAM-2).

3.1.5 capability: TBD

3.1.6 command: A request describing a unit of work to be performed by a device server. A detailed definition of a command may be found in SAM-2.

3.1.7 command descriptor block (CDB): The structure used to communicate commands from an application client to a device server. A CDB may have a fixed length of up to 16 bytes or a variable length of between 12 and 260 bytes.

3.1.8 data-in buffer: The buffer identified by the application client to receive data from the device server during the execution of a command (see SAM-2).

3.1.9 data-out buffer: The buffer identified by the application client to supply data that is sent from the application client to the device server during the execution of a command (see SAM-2).

3.1.10 device server: An object within a logical unit that executes SCSI tasks according to the rules of task management. A detailed definition of a device server may be found in SAM-2.

3.1.11 field: A group of one or more contiguous bits, a part of a larger structure such as a CDB (see 3.1.7) or sense data (see 3.1.38).

3.1.12 group object: an OSD object created in response to commands from an application client for the purpose of containing a list of user objects (see 4.4.2.1).

3.1.13 heterogeneous: A computing environment characterized by the presence of multiple computer systems, at least two of which run operating systems employing non mutually intelligible file systems.

3.1.14 host: A SCSI device with the characteristics of a primary computing device, typically a personal computer, workstation, minicomputer, mainframe computer, or auxiliary computing device or server. A host typically functions as an initiator.

3.1.15 initiator: A SCSI device containing application clients that originate device service requests to be processed in a device server. A detailed definition of an initiator may be found in SAM-2.

3.1.16 logical unit: An externally addressable entity within a target that implements a SCSI device model and contains a device server. A detailed definition of a logical unit may be found in SAM-2.

3.1.17 logical unit number (LUN): An encoded 64-bit identifier for a logical unit. A detailed definition of a logical unit number may be found in SAM-2.

3.1.18 medium: A physical entity that stores data in a nonvolatile manner (retained through a power cycle) in accordance with commands processed by the device server.

3.1.19 object: An ordered set of bytes within a storage device and associated with a unique identifier. Data is referenced by the identifier and an offset into the object. It is allocated and placed on the media by the storage device.

3.1.20 object-based storage device (OSD): A storage device in which data is organized and accessed as objects.

3.1.21 object group: A collection of user objects (see 3.1.47) with similar uses or similar attributes (see 3.1.3).

3.1.22 Object_Group_ID: the identifier for one object group (see 3.1.21).

3.1.23 object storage architecture (OSA): A term used to describe a storage architecture employing OSD.

3.1.24 object session: A set of I/O operations, subscribing to a set of previously specified quality of service characteristics, submitted by an application client to an OSD device. An object session is initiated by an OPEN on an object and terminated by a CLOSE on the object.

3.1.25 Object_Session_ID: the identifier for one object session (see 3.1.24).

3.1.26 OSD object: a root object (see 3.1.29), group object (see 3.1.12), or user object (see 3.1.47).

3.1.27 page: A regular parameter structure (or format) used by several commands. These pages are identified with a value known as a page code.

3.1.28 reset event: A protocol specific event that triggers a hard reset from a SCSI device (see SAM-2).

3.1.29 root object: a unique OSD object that is always present whose attributes contain global characteristics for the device (see 4.4.2.1).

3.1.30 SCSI block based storage device (SBC): A storage device that manages space as an ordered set of fixed length blocks. This is the typical mode used prior to the introduction of OSD.

3.1.31 SCSI device: A device that is connected to a service delivery subsystem and supports a SCSI application protocol. A detailed definition of a SCSI device may be found in SAM-2.

3.1.32 security authentication properties: TBD

3.1.33 security digital signature operators: TBD

3.1.34 security encryption operators: TBD

3.1.35 security integrity properties: TBD

3.1.36 security privacy properties: TBD

3.1.37 security secure hash operators: TBD

3.1.38 sense data: Data describing an error or exceptional condition that a device server delivers to an application client as a result of an autosense operation (see 3.1.4), asynchronous event report (see SAM-2), or REQUEST SENSE command. The format of sense data is the format defined for parameter data returned by the REQUEST SENSE command (see SPC-3).

3.1.39 sense key: The contents of the SENSE KEY field in the sense data (see 3.1.38).

3.1.40 service action: A request describing a unit of work to be performed by a device server. A service action is an extension of a command. See SAM-2 for a detailed definition of a command.

3.1.41 status: One byte of response information sent from a device server to an application client upon completion of each command. A detailed definition of status may be found in SAM-2.

3.1.42 storage device: A secondary storage unit that preserves a non-volatile copy of data sent to it and a means for retrieving any subset of that data. Discs, tapes, CD-ROM's and storage subsystems are examples of storage devices. An OSD may be any of these.

3.1.43 storage management: The undertaking of enabling, controlling and maintaining storage devices (see 3.1.42) to store, retain and deliver data. Storage management also includes selecting the appropriate storage device considering activity, cost of storage, and requirements for quality of service.

3.1.44 storage area network (SAN): A peer connection between one or more storage devices and one or more computers.

3.1.45 target: A SCSI device that receives SCSI commands and directs such commands to one or more logical units. A detailed definition of a target may be found in SAM-2.

3.1.46 task: A SCSI architecture model object within a logical unit that represents the work associated with a command or a group of linked commands. A detailed definition of a task may be found in SAM-2.

3.1.47 user object: the OSD object that contains user data (see 4.4.2.1).

3.1.48 User_Object_ID: the identifier for one user object (see 4.4.2.1).

3.1.49 vendor specific (VS): Something (e.g., a bit, field, code value, etc.) that is not defined by this standard and may be vendor defined.

3.1.50 well known object: either a root object (see 3.1.29) or a group object (see 3.1.12).

3.2 Acronyms

3DES	Triple Data Encryption Standard (see 2.2)
CDB	Command Descriptor Block (see 3.1.7)
FIPS	Federal Information Processing Standard
G	A constant equal to 4000 0000h used in describing object attribute page numbers (see 4.4.3.2)
HMAC-SHA1	Keyed-Hash Message Authentication Code - Secure Hash Standard 1 (see 2.2)
HSM	Hierarchical Storage Manager
I/O	Input/Output
ISO	Organization for International Standards

ID	Identifier
LSB	Least significant bit
LUN	Logical Unit Number (see 3.1.17)
MSB	Most Significant Bit
n/a	not applicable
INCITS	InterNational Committee for Information Technology Standards
OS	Operating System
OSA	Object Storage Architecture (see 3.1.23)
OSD	Object-based Storage Devices Commands (this standard, see clause 1)
QoS	Quality of Service
R	A constant equal to 8000 0000h used in describing object attribute page numbers (see 4.4.3.2)
RAID	Redundant Array of Independent Disks
SAM-2	SCSI Architecture Model -2 (see clause 1)
SBC	SCSI-3 Block Commands (see clause 1)
SCSI	The architecture defined by the family of standards described in clause 1
SPC-3	SCSI Primary Commands -2 (this standard, see clause 1)
VPD	Vital Product Data (see SPC-3)
VS	Vendor Specific (see 3.1.49)
XOR	exclusive-or

3.3 Keywords

3.3.1 expected: A keyword used to describe the behavior of the hardware or software in the design models assumed by this standard. Other hardware and software design models may also be implemented.

3.3.2 ignored: A keyword used to describe an unused bit, byte, word, field or code value. The contents or value of an ignored bit, byte, word, field or code value shall not be examined by the receiving SCSI device and may be set to any value by the transmitting SCSI device.

3.3.3 invalid: A keyword used to describe an illegal or unsupported bit, byte, word, field or code value. Receipt of an invalid bit, byte, word, field or code value shall be reported as an error.

3.3.4 mandatory: A keyword indicating an item that is required to be implemented as defined in this standard.

3.3.5 may: A keyword that indicates flexibility of choice with no implied preference (equivalent to "may or may not").

3.3.6 may not: A keyword that indicates flexibility of choice with no implied preference (equivalent to "may or may not").

3.3.7 obsolete: A keyword indicating that an item was defined in prior SCSI standards but has been removed from this standard.

3.3.8 optional: A keyword that describes features that are not required to be implemented by this standard. However, if any optional feature defined by this standards is implemented, then it shall be implemented as defined in this standard.

3.3.9 reserved: A keyword referring to bits, bytes, words, fields and code values that are set aside for future standardization. A reserved bit, byte, word or field shall be set to zero, or in accordance with a future extension to this standard. Recipients are not required to check reserved bits, bytes, words or fields for zero values. Receipt of reserved code values in defined fields shall be reported as error.

3.3.10 restricted: A keyword referring to bits, bytes, words, and fields that are set aside for use in other SCSI standards. A restricted bit, byte, word, or field shall be treated as a reserved bit, byte, word or field for the purposes of the requirements defined in this standard.

3.3.11 shall: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to this standard.

3.3.12 should: A keyword indicating flexibility of choice with a strongly preferred alternative; equivalent to the phrase "it is strongly recommended".

3.3.13 x or **xx:** The value of the bit or field is not relevant.

3.4 Conventions

Certain words and terms used in this standard have a specific meaning beyond the normal English meaning. These words and terms are defined either in 3.1 or in the text where they first appear. Names of commands, statuses, sense keys, and additional sense codes are in all uppercase (e.g., IDENTIFY DEVICE). Lowercase is used for words having the normal English meaning.

The names of fields are in small uppercase (e.g., STARTING BYTE ADDRESS). Normal case is used when the contents of a field are being discussed. Fields containing only one bit are usually referred to as the NAME bit instead of the NAME field.

The most significant bit of a binary quantity is shown on the left side and represents the highest algebraic value position in the quantity.

Numbers that are not immediately followed by lower-case b or h are decimal values.

Numbers immediately followed by lower-case b (xxb) are binary values.

Numbers or upper case letters immediately followed by lower-case h (xxh) are hexadecimal values.

When the value of the bit or field is not relevant, x or xx appears in place of a specific value.

Lists sequenced by letters (e.g., a-red, b-blue, c-green) show no priority relationship between the listed items. Numbered lists (e.g., 1-red, 2-blue, 3-green) show a priority ordering between the listed items.

If a conflict arises between text, tables, or figures, the order of precedence to resolve the conflicts is text; then tables; and finally figures. Not all tables or figures are fully described in the text. Tables show data format and values. Notes do not constitute any requirements for implementors.

The ISO/IEC convention of numbering is used (i.e., the thousands and higher multiples are separated by a space and a comma is used as the decimal point as in 65 536 or 0,5).

3.5 Notation for procedures and functions

In this standard, the model for functional interfaces between objects is the callable procedure. Such interfaces are specified using the following notation:

[Result =] Procedure Name (IN ([input-1] [,input-2] …]), OUT ([output-1] [,output-2] …))

Where:

Result: A single value representing the outcome of the procedure or function.

Procedure Name: A descriptive name for the function to be performed.

Input-1, Input-2, …: A comma-separated list of names identifying caller-supplied input data objects.

Output-1, Output-2, …: A comma-separated list of names identifying output data objects to be returned by the procedure.

"[…]": Brackets enclosing optional or conditional parameters and arguments.

This notation allows data objects to be specified as inputs and outputs. The following is an example of a procedure specification:

Found = Search (IN (Pattern, Item List), OUT ([Item Found]))

Where:

Found = Flag
Flag, which, if set, indicates that a matching item was located.

Input Arguments:

Pattern = … /* Definition of Pattern object */
Object containing the search pattern.

Item List = Item<NN> /* Definition of Item List as an array of NN Item objects*/
Contains the items to be searched for a match.

Output Arguments:

Item Found = Item … /* Item located by the search procedure */
This object is only returned if the search succeeds.

4 SCSI OSD Model

4.1 Overall Architecture

The OSD object abstraction is designed to re-divide the responsibility for managing the access to data on a storage device by assigning to the storage device additional activities in the area of space management. Figure 2 shows the relationship between the OSD model and a traditional SBC-based model (e.g., a file system).

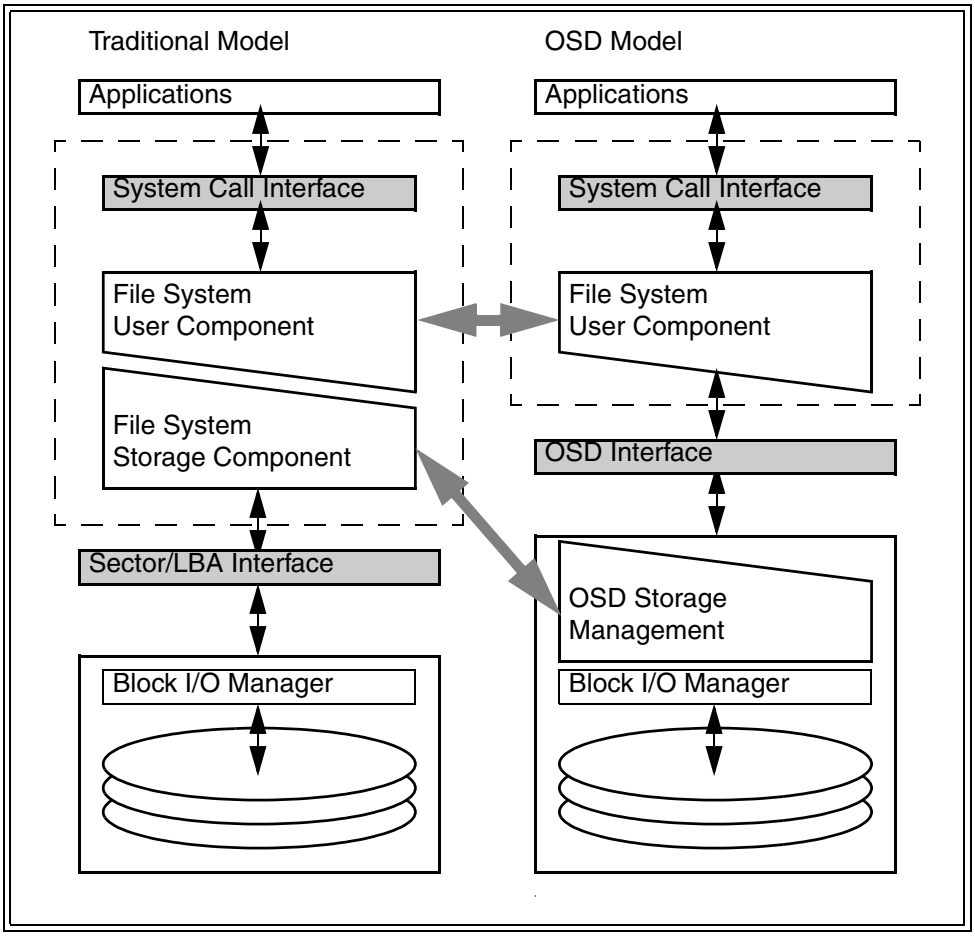

Figure 2 — Comparison of traditional and OSA storage models

The user component of the file system contains such functions as:

a) Hierarchy management;
b) Naming; and
c) User access control.

The storage management component is focused on mapping logical constructs (e.g., files or database entries) to the physical organization of the storage media. In the OSD model, the logical constructs are called user objects. OSD root and group objects provide additional mapping aids to the user component.

In addition to mapping data, the storage management component maintains other information about the objects that it stores (e.g., size, usage quotas, and associated username). For OSD objects, this information is maintained in object attributes. In the OSD model, the user component has the ability to influence the properties of object data through the specification of attributes that can, for instance, direct the location of an object to be in close proximity

to another object or to be in some part of the available space that has some higher performance characteristic (e.g., on the outer zone of a disc drive to get higher data rate).

Over the last several years many sub-components of storage management have moved to the storage device (e.g., geometry mapping, media flaw re-vectoring and media error correction). The model extends this trend to include the decisions as to where to allocate storage capacity for individual data entities and managing free space.

4.2 Elements of the example configuration

One objective of the Object Storage Architecture (OSA) is to enable the sharing of storage in a heterogeneous processor cluster. This is more complex than simply defining a new protocol. The example in this subclause illustrates how a protocol supporting the object abstraction might fit into an overall architecture.

In this example (see figure 3), the OSA architecture has three or optionally four constituents:

a) Object-Based Storage Devices (OSD Storage units),
b) Service Delivery Subsystem,
c) host systems (Initiators), and
d) optionally, a dedicated Policy/Storage Manager.

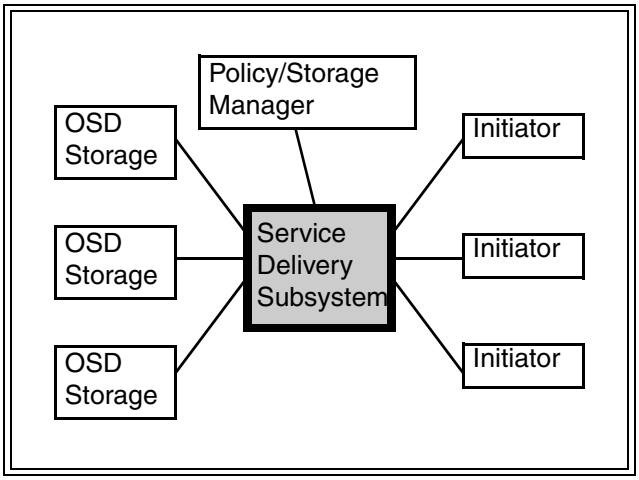

Figure 3 — Example OSA Configuration

The OSD Storage units are the storage components of the system to be shared (e.g., disc drives, RAID subsystems, tape drives, tape libraries, optical drives, jukeboxes, or other storage devices).

The application clients in multiple initiators share and directly access an OSD via the service delivery subsystem. A Policy/Storage Manager, if present, also is an application client. The service delivery subsystem is used by OSA components to intercommunicate.

A Policy/Storage Manager, if present, may perform management and security functions such as request authentication and aggregation management. The Policy/Storage Manager may relieve the other application clients of some storage management responsibilities. The Policy/Storage Manager function may be distributed among the application clients. This model ignores the role of the Policy/Storage Manager because it is not needed to describe how the commands work.

4.3 Description of the OSD Architecture

Data is stored in abstracted subsets of the available capacity on the storage device. Abstracted indicates that the data is not accessible at block or sector addresses relative to the capacity of the storage device and that the storage device allocates space for data and supplies to the application client a unique identifier that the application client uses for subsequent accesses to the data. The identifier is an unsigned integer that the storage device employs when connecting an I/O request with the data to which it applies.

It is not intended or expected that the object abstraction be a complete file system or equivalent, although it should be possible to layer a fully functioning file system on OSD commands and constructs. Notions such as streams, file naming, complex hierarchical relationships, and complex ownership and access control rules are assumed to be part of a application clients accessing an OSD or collection of OSDs, not part of an OSD itself.

In the context of a SCSI logical unit (see SAM-2), an OSD provides the additional objects listed in table 1 as the abstractions upon which a file system layer may be built.

Table 1 — OSD Specific Model Objects

OSD Objects Representing Stored Data		**OSD Model Objects Representing Transient Application Client Activities**	
OSD Object	**Reference**	**Model Object**	**Reference**
Root Object	4.4.2.2	Policy	
Group Object	4.4.2.2	Object Session	4.4.4.2
User Object	4.4.2.5		
Associated Data	**Reference**		
Attributes	4.4.3		

Editors Note 1 - ROW: Table 1 attempts to distill the OSD specific model objects from r04 figure 6 into something that can act as a guide to the rest of the OSD model. It should be viewed as a work in progress. Concerns exist about calling Policy, Object Session, and Parameters objects and new terminology is likely in future revisions.

4.4 Principal OSD Features

4.4.1 The OSD device type

An OSD device is a logical unit that returns the OSD device type value in response to an INQUIRY command. An OSD device contains only objects not logical blocks. All objects have attributes (see 4.4.3) associated with them.

4.4.2 Stored data objects

4.4.2.1 Stored data object types

There are three different types of stored data objects:

a) **Root**: This unique object is always present in the device. Its attributes (see 4.4.3) contain global characteristics for the device. This includes the total capacity of the logical unit, maximum number of objects that it may contain, as well as certain quality of service characteristics such as data integrity characteristics (e.g.,

the device stores all its data in RAID5). Its data contains the list of currently valid Object_Group_IDs. The root object is maintained by the OSD.

b) **Group**: This object is created by specific commands from an application client and represents an object group. Its purpose is to contain a list of user objects that share some common attributes. Its attributes include the current number of user objects, the quota capacity of the object group, and the current capacity utilized by the object group. The default attributes for a group object are inherited from the attributes in the root object. The data component of an object group is the list of currently valid User_Object_IDs. The group object is maintained by the OSD.

c) **User**: These are the primary objects that contain file or database data. The data for a user object is user data; the OSD manages this data on behalf of the initiators. The attributes for a user object include the logical size of the user data, and quality of service attributes. Default attributes for a user object are inherited from the attributes of the group object in which it is listed.

When an OSD device is shipped from the factory, there may only be a root object whose attributes are the factory defaults. There may be no group or user objects, with the result that the data for the root object is empty. The FORMAT command shall restore an OSD logical unit to this original state.

4.4.2.2 Well known objects

There shall be one well known object describing each OSD logical unit, called the root object. Every object group shall be represented by a well known object, called the group object. These objects serve as landmarks for an application client navigating the OSD organization.

For the root object, the User_Object_ID shall be zero and the Object_Group_ID shall be zero. For the group object, the Object_Group_ID is assigned by the OSD when the object group is created, and the User_Group_ID shall be zero. These assignments are summarized in table 2.

Table 2 — Well Known Objects

Well Known Object	Object_Group_ID	User_Object_ID
Root	0	0
Group	N [a]	0
[a] N shall not be equal to zero.		

To get a list of the valid Object_Group_IDs, an application client may send LIST (see 6.12) service action request to the device server specifying the root object. To get a list of the User_Object_IDs in an object group, the application client may send a LIST service action to the device server specifying the Object_Group_ID of the group object for which the list is desired.

The device server shall terminate all READ, WRITE and APPEND service actions sent to well known objects ~~the root object or group object~~ with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to INVALID FIELD IN CDB.

4.4.2.3 Root object

There is only one root object per OSD logical unit that may be addressed as both Object_Group_ID and User_Object_ID equal to zero. The root object acts as a starting point for navigation of the structure on an OSD logical unit.

4.4.2.4 Object groups

User objects are collected into groups, called object groups each one of which is represented by a group object. There may be many group objects, up to the capacity of the OSD.

~~The object group with the Object_Group_ID zero is the default object group. The default object group shall be created as part one of the operations performed by the FORMAT OSD service action (see 6.8). A CREATE OBJECT GROUP service action (see 6.6) or REMOVE OBJECT GROUP service action (see 6.16) that specifies the default object group shall be terminated with a CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.~~

An Object_Group_ID uniquely identifies each object group. Object_Group_ID values are assigned as shown in table 3.

Table 3 — Object_Group_ID value assignments

Object_Group_ID	Description
0h	Root object
1h - Fh	Reserved
10h - FFFF FFFFh	Assigned by the OSD

To get a list of the valid Object_Group_IDs, an application client may send LIST service action (see 6.12) request to the device server specifying the root object.

Optionally, there may be a capacity quota associated with an object group, requiring the OSD to ensure that the sum of storage capacity occupied by all the user objects in an object group does not exceed the capacity quota. The storage capacity occupied by the group object representing the object group shall be counted against the capacity quota established for the object group.

The OSD device may be instructed to reject WRITE, CREATE, CREATE AND WRITE, or APPEND service actions that would result in an object group consuming more storage capacity than it has been assigned. The capacity quota lets several independent application clients create and store data in user objects without the danger of any consuming more than an allocated percentage of the available capacity.

When an object group is created using the CREATE OBJECT GROUP service action (see 6.6), a group object shall be built to act as the point of departure for navigating through the user objects in the object group. The group object representing the object group shall have User_Object_ID zero and shall have the Object_Group_ID assigned to the object group by the OSD.

~~The group object representing the object group contains a list of the User_Object_ID's for all user objects belonging to the object group.~~

4.4.2.5 User objects

User objects contain arbitrary data (i.e., the content of this data is owned by the applications that cause the creation, writing and reading the user objects). User objects have the Object_Group_ID of the object group to which they belong and an User_Object_ID that is assigned by the OSD when the user object is created.

The combination of Object_Group_ID and User_Object_ID uniquely identifies the root and each group or user object. Combined Object_Group_ID and User_Object_ID values are assigned as shown in table 4.

Table 4 — Object_Group_ID and User_Object_ID value assignments

Object_Group_ID	User_Object_ID	Description
0h	0h	Root object
0h	1h - FFFF FFFF FFFF FFFFh	Reserved
1h - Fh	0h - FFFF FFFF FFFF FFFFh	Reserved
10h - FFFF FFFFh	0h	Object group (assigned by OSD)
10h - FFFF FFFFh	1h - Fh	Reserved
10h - FFFF FFFFh	10h - FFFF FFFF FFFF FFFFh	User object (assigned by OSD)

The READ, WRITE and APPEND service actions read and modify the data in a user object. Four fields are needed to identify the data in a user object to be transferred (see table 5).

Table 5 — Addressing data in a user object

Field	Description	Required By		
		READ	WRITE	APPEND
OBJECT_GROUP_ID	Specifies the object group in which the object is listed	Y	Y	Y
USER_OBJECT_ID	Specifies the object within the object group named in the OBJECT_GROUP_ID field	Y	Y	Y
STARTING BYTE ADDRESS	Specifies the starting byte location within the object for the data transfer	Y	Y	N
LENGTH	Specifies the number of bytes to be transferred	Y	Y	Y

Editors Note 2 - ROW: A request has been made to remove table 5 and the paragraph that introduces it. The editor is inclined to honor the request. Therefore, table 5 will be removed in revision 7 unless objections are heard.

4.4.3 Object attributes

4.4.3.1 Object attributes overview

File systems and other SBC-based systems store not only user data but data about the user data, sometimes called metadata. OSD object attributes allow the association of metadata with any OSD object (i.e., root, group, or user). Object attributes may be used to describe specific characteristics of an object. This includes the total amount of bytes occupied by the object (including attributes), logical size of the object, the time the object was last modified, and many other parameters.

Any OSD service action may retrieve object attributes and any OSD service action may store object attributes. In addition, the GET ATTRIBUTES service action (see 6.10) and SET ATTRIBUTES service action (see 6.18) allow object attributes to be retrieved and store without performing other OSD operations.

Object attributes are organized in pages for identification and easy reference. The attributes within a page have similar sources or uses. Within each object attributes page, object attributes are identified by an attribute number. Each object attributes pages is associated with one of the following:

a) The root object;
b) A group object; or
c) A user object.

The same object page shall not be associated with more than one OSD object.

The structures of object attribute pages are defined by standards (e.g., this standard, other American National Standards, ISO standards), by OSD applications specifications (e.g., Linux, Windows, and Oracle), by publicly available manufacturer product specifications, and by other written documentation. A range of vendor specific object attribute pages is defined for which there may be no publicly available structure description.

4.4.3.2 Object attribute pages

Each object attribute page contains object attributes with similar sources or uses. Identifying numbers are assigned to object attribute pages with ranges of page numbers indicating the type of object with which an object attributes page is associated as shown in table 6.

Table 6 — Object attribute page numbers

Page number	OSD object type with which the object attributes page is associated
0h - 3FFF FFFFh	User
4000 0000h - 7FFF FFFFh	Group
8000 0000h - BFFF FFFFh	Root
C000 0000h - FFFF FFFEh	Reserved
FFFF FFFFh	All attributes pages

To simplify the description of object attributes pages associated with group objects or the root object, two constant values are used in this standard:

a) G is equal to 4000 0000h; and
b) R is equal to 8000 0000h.

Using these constants an object attributes page associated with an object group is described as having an object attributes page number of, for example, G+5h, meaning that the written out object attributes page number is 4000 0005h. Similarly, an object attributes page number of R+Ah represents an object attributes page that is associated with the root object and has a written out object attributes page number of 8000 000Ah. No constant is needed for object attributes pages that are associated with user objects.

The ranges of object attribute page numbers shown in table 6, are subdivided as shown in table 7.

Table 7 — Object attribute page number sets

Page number within range	Description	Assignment
0h - 7Fh	Defined by this standard	Fixed [a]
80h - EFFFh	Defined by other standards	Fixed [a]
F000h - FFFFh	Defined by OSD manufacturer product specifications	Fixed [a]
1 0000h - 2FFF FFFFh	Assigned by the OSD logical unit	Dynamic [b]
3000 0000h - 3FFF FFFFh	Vendor specific	n/a

[a] The fixed assignment of object attribute page numbers means that the document describing the object attribute page also specifies the object attribute page number to be used by the page.

[b] The object attribute page numbers set aside for dynamic assignment are those from which the CREATE ATTRIBUTE PAGE service action (see 6.6) will select when no specific object attribute page number is requested.

Object attribute pages shall contain:

a) Object attributes (see 4.4.3.3); or
b) References to object attributes in other object attribute pages.

If an object attribute page contains one reference to an object attribute in another object attribute page, all object attributes in that page shall be references to object attributes in another object attribute page, with the exception of the object attribute with the object attribute number 0h that shall contain the name of the object attributes page as described in 4.4.3.3.

If an object attribute references an object attribute in another object attributes page, the referenced object attribute shall not be a reference to another object attribute (i.e., only one level of reference indirection is allowed).

See 7.1.2 for information about attributes pages defined by this standard. For an example of an attributes page containing object attributes see 7.1.2.13. For an example of an attributes page containing references to object attributes in other object attributes pages see 7.1.2.18.

4.4.3.3 Object attributes

Each object attribute within an object attributes page (see 4.4.3.2) has a unique number between 0h and FFFF FFFFh. The description of each object attribute defines the format and usage of that attribute. For examples of object attribute definitions see 7.1.2.

Object attributes page pages contain object attributes each of which is assigned an attribute number. The object attribute with the attribute number 0h shall contain the name of the page in the format described in 7.1.2.2. The object attribute number FFFF FFFFh shall represent all object attribute values in the page, not an object attribute value.

4.4.3.4 Object attributes directories

OSD root, group, and user objects have associated attributes directory pages as defined in table 8.

Table 8 — Object attributes directory pages

Page Number	Page Name	Attributes page contents	Support Requirement	Reference
R+0h	Root Directory	Contains one object attribute for every object attribute page known to the OSD logical unit.	Mandatory	7.1.2.3
G+0h	Group Directory	Contains one object attribute for every object attribute page associated with the group object or any user object in the group.	Optional	7.1.2.4
0h	User Object Directory	Contains one object attribute for every object attribute page associated with the user object.	Optional	7.1.2.5

Attributes directory pages shall be maintained by the OSD.

Application clients may modify an attributes directory page only by modifying the contents of object attribute 0h in an object attributes page other than the attributes directory page. Any command that attempts to modify an attributes directory page in any other manner shall be terminated with a CHECK CONDITION status. The sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to INVALID FIELD IN CDB or INVALID FIELD IN PARAMETER LIST as appropriate.

4.4.4 Application client activities objects

4.4.4.1 Policy

TBD

4.4.4.2 OSD object sessions

An object session is a set of state information maintained in the OSD for the purposes of setting parameters for data transfer. Object sessions are used only for READ (see 6.14), WRITE (see 6.19), and APPEND (see 6.2) service actions. Except for the default object session, Object sessions are not persistent across a reset event or a power off cycle.

Every OSD logical unit has a default object session that determines the parameters of its underlying data transfer machinery. The Object_Session_ID of the default object session is zero.

Optionally, an OSD device may support other object sessions. The Object_Session_ID of any object session other than the default object session shall be non-zero. The Object_Session_IDs are generated by the OSD and provided to the application client via the Object_Session_ID attribute in the Current Command attributes page (see 7.1.2.15). The parameters that govern an object session may be queried using the GET ATTRIBUTES (see 6.10) service action and changed using the SET ATTRIBUTES (see 6.18) service action.

READ, WRITE, and APPEND service actions shall be processed only under an existing object session. That is, such actions shall not create a object session, they shall only fall under the auspices of an existing one.

A CLOSE (see 6.3) service action is used to close one or all non-default object sessions.

The minimum number of object sessions supported shall be one (i.e., the default object session). The maximum number of object sessions supported is vendor specific.

Once a object session is established, READ, WRITE and APPEND service actions may be requested within the context of that object session.

One parameter of a object session may be an expiration time. Object sessions with an expiration time parameter may close without any specific service action being requested by the application client. The closing of an object session or the change of parameters for a object session shall not affect the data transfer for any read/write operation already being processed by the device server within that object session. Only new read/write operations specifying the affected Object_Session_ID shall rejected.

4.4.5 Security

4.4.5.1 Security Introduction

A complete and useful OSD environment should:

a) Authenticate application client identities,
b) Test each requested operation against the environment's database of authorized transformations,
c) Ensure that the integrity of each operation is not tampered with during transmission,
d) Protect the privacy of data, access patterns and application clients,
e) Identify the source of inappropriate activity, and
f) Survive inappropriate activity with minimal degradation of service qualities.

Proper operation of some of these security features requires support from the SCSI protocol and as such is beyond the scope of this standard.

Security is the confident state of mind of users of a OSD environment that exhibits compelling instances of these properties. While it is possible that application clients can be trusted to accurately state their identities and authorizations, and it is possible for the service deliver subsystem and other initiators to be trusted not to interfere with the integrity and privacy of all issued requests so that inappropriate activity never occurs, this standard is designed not to depend upon such high degrees of trustworthiness.

To avoid specialization to any particular file system or server application, OSD security mechanisms are expected to enforce a wide range of possible authentication and access policies whose details may not be determined until OSD systems are available to the software designers, users and administrators of a specific application system. Thus, the details of access control policy as well as user authentication and authorization are beyond the scope of this standard.

OSD security defines encryption mechanisms enabling OSDs to protect privacy and digital signature mechanisms enabling OSDs to enforce authentication, authorization, integrity, auditing and isolation policies set by customer decisions through policy/storage managers without unnecessary limitation on the flexibility of those policies.

4.4.5.2 OSD security key hierarchy

A basic theme in cryptographic security is the controlled sharing of secrets. Fundamentally, to perform an OSD access, an application client demonstrates that either it shares a secret with the OSD device, or that the access was previously authorized by a Policy/Storage Manager who shares a secret with the OSD device. Client level key management is a critical security function, but since it is inherently a system wide function, it is properly a function of a higher level than the storage system. Client level keys should be distributed privately to application clients by

Policy/Storage Managers, valid for a lifetime that is inversely proportional to their frequency of use, and protected against disclosure by a device's owner and users.

Policy/Storage Managers, some owning OSDs and allocating them to other Policy/Storage Managers (for example, a backup system may own devices and allocate them to file systems or database systems), need to share secrets with OSDs. To protect against any specific key being overused and exposed to yet-to-be-created key discovery algorithms, keys should have a primary and secondary version. Secondary keys should more commonly used than primary keys and should be changed frequently while primary keys should be rarely used to enable their lifetime to be much longer. Since OSD resources are shared among non-cooperating Policy/Storage Managers by granting each an object group which behaves like a virtual OSD, each object group needs its own set of keys. Finally, to avoid the simultaneous revocation of all signatures made by a working (secondary) key that is due to change, working keys come in pairs, both valid, but only one being used to generate new signatures. This way old signatures are not necessarily revoked by the first working key change; until the second working key change these signatures may still be valid.

Editors Note 3 - ROW: This appears to be saying that the group object has been overloaded with a security function that more properly should be defined as a fourth object type.

The hierarchy of security keys is shown in table 9.

Table 9 — OSD security key hierarchy

Name	Description
Master key	OSD owner's key; preferably used only at installation
Device key	Frequently changed online proxy for owner's key
Object group key	Less frequently changed online per object group key
Object group working key A	Frequently changing, commonly used per object group key A
Object group working key B	Frequently changing, commonly used per object group key B

Initially an OSD device has a single key preinstalled. This is known as the master key. The master key may only be modified by a command encrypted with the previous value of the master key. While an OSD has no control over the environment into which its owner modifies its master keys, it is recommended that this key only be changed when the OSD device is attached to a physically secured private network without any gateway to application client systems (i.e., on a two-port network on the desk of an administrator).

The device key is set infrequently using the master key or the previous value of the device key. This key exists to provide a more frequently used variant of the master key. With this key, the master key need only be used in operations that modify this key. This helps protect the master key from attack.

Each object group has a unique object group key. An object group key is set at object group creation time, and is modified with the device key or the previous value of the object group key. In this manner, two file system managers utilizing different OGs on the same OSD may be provided separate object group keys, and thus be totally denied access to one anothers' OGs. Each object group also has two object group working keys, A and B that are set using the object group key. As with the master and device keys, the working keys are intended to limit the use of the object group key.

Most operations that require the use of a key in the key hierarchy should use object group working keys, and Policy/Storage Managers should change the object group working keys regularly (e.g., daily). Two object group working keys are provided in each object group because OSD capabilities, authenticated by a working key signature, are reusable over an extended (e.g., 12 hour) time period. By having a second object group working key to use in future capability generation, the first of these object group working keys may be changed without invali-

dating capabilities signed with the other. This allows capabilities generated shortly before a working key is changed to continue to work until they gracefully timeout according to their expiration time.

4.4.5.3 Digital signatures

A digital signature is an encrypted value the successful decryption of which verifies that:

a) A collection of data fields contain correct values; and
b) The values in those data fields were prepared by the entity that created the digital signature.

The preparation of a digital signature involves two steps:

1) The a hash function is applied to the collection of data fields that are to be protected by the digital signature; and
2) The value resulting from the application of the hash function is encrypted using a key known only to the entity responsible for the digital signature.

The hash and encryption functions required by OSD are shown in table 10.

Table 10 — OSD Security Hash and Encryption Functions

Security Function	OSD Requirement	Reference
Hash	HMAC-SHA1	FIPS 180-1 (1995)
Encryption	3DES	ANSI X9.52-1998

OSD Policy/Storage Managers use digital signatures to provide these protections to capabilities parameters (see 4.4.5.4).

The key used in a digital signature is not available to the application client but is available to both the Policy/Storage Manager and the OSD. Although an application client is not capable of generating this hash, it may hold the hash and use it to prove to the OSD device that the Policy/Storage Manager granted the associated capability. Because an application client is not capable of generating a correct hash and because the correct hash includes the capability fields as well as the secret key, an application client that has been granted a specific capability is not capable of changing any values in the capability's fields to gain inappropriate actions from an OSD device.

4.4.5.4 OSD capabilities

Application clients except for Policy/Storage Managers should never possess one of the keys in the OSD key hierarchy. Instead, they should be issued individual capabilities by Policy/Storage Managers. A capability describes the accesses that application clients are allowed to perform by proving that they have been given that capability.

Capabilities are validated by means of a digital signature (see 4.4.5.3). A central feature of this OSD capability system is that these signed hashes may themselves be treated as a key, provided that the Policy/Storage Manager has distributed them securely. By using the signed hash distributed to it as a secret in the computation of a derived signed hash, an application client is capable of proving to an OSD device that it holds the first signed hash. Hence we call these signed hashes capability key-signatures to emphasize their roles as keys. A signed capability is the union of a capability and its capability key-signature.

The information contained in a capability (e.g., which OSD service actions are permitted by the capability) is detailed in 5.1.2.1.1.

~~4.4.5.5 OSD Time~~

~~OSD devices are required to record the time at which certain events happen or will happen (e.g., the time a user object is created, and the time capabilities parameters will expire). This time shall be stored in an 8-byte unsigned integer as the number of 1-millisecond ticks since 00:00 o'clock, November 17, 1858 (i.e., the Smithsonian base date and time for the astronomic calendar). A stored OSD time shall reflect the time the event occurred or will occur to at least the nearest half second, and should reflect the time to the nearest tenth of a second.~~

4.5 Overview of OSD Operation

4.5.1 Preparing a device for OSD operation

Editors Note 4 - ROW: No attempt has been made to make this example consistent with the command set and attributes changes made in OSD revision 06.

In order for a logical unit to accept and process OSD commands, it shall have been initialized as an OSD device. An application client issues the commands in to initialize and OSD device.

Table 11 — OSD initialization sequence

Action	Parameters	Description
SET KEY	Master, Device	Initialize higher OSD keys
FORMAT OSD	Length (optional)	Construct OSD control structures
CREATE OBJECT GROUP	Capacity Quota (optional)	Initialize object group in which user objects may be created
SET KEY	Object Group Key, Working Keys A & B	Initialize object group keys

Upon completion of these commands the storage device is an OSD device with security established. The device server shall accept OSD mandatory commands and may accept OSD optional commands.

4.5.2 Startup — discovery and configuration

Editors Note 5 - ROW: This whole subclause is not SCSI. In SCSI initiators discover targets not the other way around. Finally, as the first sentence says, “configuration techniques are a function of the interconnect protocols.” This standard is a command set standard, not a protocol standards. This subclause flat out does not belong in OSD.

Startup, discovery, and configuration techniques are a function of the interconnect protocols. When the OSA device is powered up the OSD device shall identify itself to all initiators or to a common point of reference, such as a name service on the interconnect. The details of the discovery process is beyond the scope of OSD. For example, in a Fibre Channel fabric based OSA, the OSD devices, Policy/Storage Manager (if any), and other application clients would log onto the fabric. From the fabric they may learn of the existence of all other OSA components. They may use these fabric services to identify all other components. The application clients learn of the existence of the OSD devices they may have access to, while the OSD devices may learn where to go when they need to locate another storage device. Similarly the Policy/Storage Manager (if any) learns of the existence of OSD devices from the fabric services.

Optionally each OSA component may identify to the Policy/Storage Manager any special configuration information. Storage device level service attributes may be communicated once to the Policy/Storage Manager, where all other components may learn of them. For example an application client may need to be informed of the introduction of additional storage subsequent to startup, noted by an attribute set when the application client logs onto the Policy/Storage Manager. The Policy/Storage Manager may do this automatically whenever a new OSD device is added to the configuration, including conveying important characteristics, such as it being RAID 5, mirrored, etc.

4.5.3 Verification

The OSD device shall reject invalid I/O requests with a CHECK CONDITION status and a sense key of ILLEGAL REQUEST.

Editors Note 6 - ROW: I am not sure what is left of this subclause belongs here.

4.5.4 Example of accessing data on an OSD

Editors Note 7 - ROW: No attempt has been made to make the examples in this subclause consistent with the command set and attributes changes made in OSD revision 06.

File system function is beyond the scope of this standard, but for the sake of illustration a simple PC/UNIX-like file system is assumed in this example. The file system consists of a single file in a single subdirectory:

/father/son

Where "father" is the name of the directory to be created and "son" is the file.

Table 12 lists the sequence of OSD commands that may result in the file system being created. It is assumed that the OSD device and the object group are known and that the application client holds a valid signed capability for each object accessed.

Table 12 — OSD command sequence for creating a file

Step	Service Action	Object_ Group_ID	User_ Object_ID	Discussion
1	GET ATTRIBUTE	n	1	Get User_Object_ID of the root object (r).
2	READ	n	r	Make sure "father" does not already exist.
3	CREATE	n		Returns User_Object_ID (s), to hold file "son".
4	CREATE	n		Returns User_Object_ID (f), to hold directory "father".
5	WRITE	n	s	Write contents of "son" - one or more WRITEs involved
6	WRITE	n	f	Write contents of "father" - one or WRITEs involved
7	WRITE	n	r	Root directory revised to contain "father"

Editors Note 8 - ROW: I do not understand the need to get the User_Object_ID of the root object. As per 4.4.2.2, the User_Object_ID of the root object is zero. I do not see anything in 4.4.2.1 to suggest that revising the root object is necessary or appropriate at this point. Furthermore, 4.4.2.2 explicitly says that a WRITE to the root object is an error.

The CREATE AND WRITE service action is capable of transferring data to the newly created object. As is shown in table 13, separate WRITEs are not need to fill "son" or "father" with data when this option is used.

Table 13 — OSD command sequence using CREATE AND WRITE

Step	Service Action	Object_ Group_ID	User_ Object_ID	Discussion
1	GET ATTRIBUTE	n	1	Get User_Object_ID of root object (r).
2	READ	n	r	Make sure "father" does not already exist.
3	CREATE AND WRITE	n		Returns User_Object_ID (s), that holds file "son".
4	CREATE AND WRITE	n		Returns User_Object_ID (f), that holds directory "father".
5	WRITE	n	r	Root directory revised to contain "father"

Editors Note 9 - ROW: I have the same problems here that I have with table 12.

Table 14 is an example that includes OPEN and CLOSE actions. These may be used to "lock" the root directory while it is being updated with the new directory "father".

Table 14 — OSD command sequence with OPEN and CLOSE actions

Step	Service Action	Object_ Group_ID	User_ Object_ID	Discussion
1	GET ATTRIBUTE	n	1	Get User_Object_ID of root object (r).
2	OPEN	n	r	Returns Object_Session_ID, QoS is LOCK this object
3	READ	n	r	Make sure "father" does not already exist.
4	CREATE	n		Returns User_Object_ID (s), to hold file "son".
5	CREATE	n		Returns User_Object_ID (f), to hold directory "father".
6	WRITE	n	s	Write contents of "son" - one or more WRITEs involved
7	WRITE	n	f	Write contents of "father" - one or WRITEs involved
8	WRITE	n	r	Root directory revised to contain "father"
9	CLOSE	n	r	Unlock root directory

Editors Note 10 - ROW: In addition to all the problems I have table 12, the OPEN in this example appears to be locking a user object in object group n. Unless maybe the root object and root directory are two different things. That would be okay but then step1 must be getting the User_Object_ID of the root directory for object group n, not the User_Object_ID of the root object.

4.5.5 OSD mandatory actions template

Table 15 lists the action codes mandatory for OSD devices.

Table 15 — OSD Mandatory commands with field lengths

Service Action	Field with size in bytes (blank means field not used)							
	Options	Object_ Group_ID	User_ Object_ID	Object_ Session_ ID	Starting Byte	Byte Length	Attribute Mask	Reserved
FORMAT OSD	2					8 [b]		4
CREATE OBJECT	2	4		4 [a]	8	8	8 [a]	
OPEN [c]	2	4	8	4 [a]			8 [a]	
READ	2	4	8	4 [a]	8	8		
WRITE	2	4	8	4 [a]	8	8		
APPEND [d]	2	4	8	4 [a]		8		
FLUSH OBJECT	2	4	8					
CLOSE [c]	2	4	8	4 [a]				
REMOVE	2	4	8					
CREATE OBJECT GROUP	2							
REMOVE OBJECT GROUP	2							
SET ATTRIBUTES [e]	2	4	8	4 [a]			8	
GET ATTRIBUTES [e]	2	4	8	4 [a]			8	

[a] This field has an optional meaning depending on a specified option.

[b] The length field on the FORMAT OSD action contains the amount of capacity to be allocated to the OSD.

[c] OPEN and CLOSE an object mark the beginning and end object session composed of a set of actions requiring additional management support or quality of service.

[d] APPEND is a write command with no starting byte. The OSD determines the last byte of an object and puts the data accompanying the command at the end of the object, then updates the object logical length. This command allows multiple application clients to contribute to a log file without each in turn having to gain control of the object and lock it from access by others.

[e] The GET ATTRIBUTE and SET ATTRIBUTE actions are like MODE SENSE and MODE SELECT commands for the OSD environment in that they are used to interrogate and supply operating mode attributes for objects, OGs, and OSD themselves.

4.5.6 OSD optional actions template

Table 16 lists the action codes optional for OSD devices.

Table 16 — OSD Optional commands with field lengths

Service Action	Field with size in bytes (blank means field not used)							
	Options	Object_ Group_ID	User_ Object_ID	Object_ Session_ ID	Starting Byte	Byte Length	Source OSD	Source Object_ Group_ID
IMPORT OSD [a]	2	4	8	4	8	8	16	4
[a] The IMPORT OBJECT command enables an application client to instruct an OSD device to read an object from a second OSD creating and writing a copy onto itself.								

Editors Note 11 - ROW: Is SET KEY mandatory or optional? Does it belong in this subclause or the previous one?

4.6 Reservations

The access enabled or access disabled condition determines when an application client may store or retrieve user data on all or part of the medium. Access may be restricted for read operations, write operations, or both. This attribute may be controlled by an external mechanism, by object session parameters or by the PERSISTENT RESERVE IN and PERSISTENT RESERVE OUT commands (see SPC-3). The OSD devices shall not support the RESERVE and RELEASE commands.

The PERSISTENT RESERVE IN and PERSISTENT RESERVE OUT commands define how different types of restricted access may be achieved, and to whom the access is restricted. This subclause describes the interaction of the application client that requested the reservation, and the other application clients.

An application client uses reservations to gain a level of exclusivity in access to all or part of the medium for itself or another application client. It is expected that the reservation is retained until released. The device server ensures that the application client with the reservation is able to access the reserved media within the operating parameters established by that application client.

Reservation restrictions are placed on commands as a result of access qualifiers associated with the type of reservation. The details of commands that are allowed under what types of reservations are described in table 17.

Commands from initiators holding a reservation should complete normally. The behavior of commands from registered initiators when a registrants only or all registrants persistent reservation is present is specified in table 17.

An unlinked command shall be checked for reservation conflicts before the task containing that command enters the enabled task state. The reservation state as it exists when the first command in a group of linked commands enters the enabled task state shall be used in checking for reservation conflicts for all the commands in the task. Once a task has entered the enabled task state, the command or commands comprising that task shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation. Any command in a group of linked commands that changes the reservation state shall be the last command in the group.

For each command, this standard or SPC-3 defines the conditions that result in RESERVATION CONFLICT.

Table 17 — OSD commands that are allowed in the presence of various reservations

Command	Addressed LU has this type of persistent reservation held by another initiator				
	From any initiator		From registered initiator (RR all types)	From initiator not registered	
	Write Excl	Excl Access		Write Excl RR	Excl Access – RR
APPEND	Conflict	Conflict	Allowed	Conflict	Conflict
CLOSE	Allowed	Conflict	Allowed	Allowed	Conflict
CREATE	Conflict	Conflict	Allowed	Conflict	Conflict
CREATE AND WRITE	Conflict	Conflict	Allowed	Conflict	Conflict
CREATE ATTRIBUTES PAGE	Conflict	Conflict	Allowed	Conflict	Conflict
CREATE OBJECT GROUP	Conflict	Conflict	Allowed	Conflict	Conflict
FORMAT OSD	Conflict	Conflict	Allowed	Conflict	Conflict
FLUSH OBJECT	Conflict	Conflict	Allowed	Conflict	Conflict
GET ATTRIBUTES	Allowed	Conflict	Allowed	Allowed	Conflict
IMPORT USER OBJECT	Conflict	Conflict	Allowed	Conflict	Conflict
LIST	Allowed	Conflict	Allowed	Allowed	Conflict
OPEN	Allowed	Conflict	Allowed	Allowed	Conflict
READ	Allowed	Conflict	Allowed	Allowed	Conflict
REMOVE	Conflict	Conflict	Allowed	Conflict	Conflict
REMOVE ATTRIBUTES PAGE	Conflict	Conflict	Allowed	Conflict	Conflict
REMOVE OBJECT GROUP	Conflict	Conflict	Allowed	Conflict	Conflict
SET ATTRIBUTES	Conflict	Conflict	Allowed	Conflict	Conflict
~~SET KEY~~	~~Conflict~~	~~Conflict~~	~~Allowed~~	~~Conflict~~	~~Conflict~~
OSD SYNCHRONIZE CACHE	Conflict	Conflict	Allowed	Conflict	Conflict
WRITE	Conflict	Conflict	Allowed	Conflict	Conflict
Key: **LU**=Logical Unit, **Excl**=Exclusive, **RR**=Registrants Only or All Registrants					

5 Common Formats

5.1 Command format

5.1.1 OSD CDB format

The OSD CDB comprises a 10-byte header that shall not be encrypted, followed by a body section that may either be encrypted or not depending on the content of a controlling field in the header.

An application client conveys a command CDB to the device server. If a device server receives a CDB containing an operation code that is invalid or not supported, it shall return CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and an additional sense code of INVALID COMMAND OPERATION CODE.

The OSD commands defined in this standard use the variable length CDB format (see SPC-3). In the variable length CDB used by OSD commands (see table 18), an OPERATION CODE field containing 7Fh is the first byte and a CONTROL byte is the second byte. The general structure of the OPERATION CODE field and CONTROL byte are defined in SAM-2.

Table 18 — Typical OSD CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (7Fh)							
1	CONTROL							
2	Reserved							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved		IS_CDB	IS_DATA	Reserved		PS_CDB	PS_DATA
7	ADDITIONAL CDB LENGTH (n-7)							
8	(MSB)			SERVICE ACTION				
9								(LSB)
10	Service action specific fields							
n								

The IS_XXX fields specify the types of information integrity security to be applied to the command and data. The PS_XXX fields specify the types of information privacy security to be applied to the command and data.

Editors Note 12 - ROW: The use of bytes 2 through 6 inclusive needs to be coordinated with SPC-3.

An Integrity Security for CDB (IS_CDB) bit of zero specifies that the device server shall not require the service delivery subsystem to perform information integrity security operations on the CDB and response data. An IS_CDB bit of one specifies that the device server shall require the service delivery subsystem to perform information integrity security operations on the CDB and response data. If the IS_CDB bit is one and the service delivery subsystem is unable to perform information integrity security operations on the CDB and response data, the

command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

An Integrity Security for Data (IS_DATA) bit of zero specifies that the device server shall not require that the service delivery subsystem perform information integrity security functions on the data transfers associated with the service action. An IS_DATA bit of one specifies that the device server shall require the service delivery subsystem to perform information integrity security operations on the data transfers associated with the service action. If the IS_DATA bit is one and the service delivery subsystem is unable to perform information integrity security operations on the data transfers associated with the service action, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

An Privacy Security for CDB (PS_CDB) bit of zero specifies that the device server shall not require the service delivery subsystem to perform information privacy security operations on the CDB and response data. An PS_CDB bit of one specifies that the device server shall require the service delivery subsystem to perform information privacy security operations on the CDB and response data. If the PS_CDB bit is one and the service delivery subsystem is unable to perform information privacy security operations on the CDB and response data, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

An Privacy Security for Data (PS_DATA) bit of zero specifies that the device server shall not require that the service delivery subsystem perform information privacy security functions on the data transfers associated with the service action. An PS_DATA bit of one specifies that the device server shall require the service delivery subsystem to perform information privacy security operations on the data transfers associated with the service action. If the PS_DATA bit is one and the service delivery subsystem is unable to perform information privacy security operations on the data transfers associated with the service action, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

When the service action has associated capabilities (i.e., when the CAPABILITIES LENGTH field contains a non-zero value), the PS_DATA bit may be set to one to cause the capabilities information to be transferred under privacy security.

NOTE 2 The only time the setting of the IS_CDB, IS_DATA, PS_CDB, and PS_DATA bits place requirements on the security actions of the service delivery subsystem is when one or more of them are one. When any of the bits are zero the service delivery subsystem may still perform any the identified security functions.

The ADDITIONAL CDB LENGTH field specifies the number of bytes following it in the variable length CDB. If the value in the ADDITIONAL CDB LENGTH field is less than the length defined for the service action by this standard, all beyond the specified CDB length shall be processed as if they contain zero.

The command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB if the value in the ADDITIONAL CDB LENGTH field results in truncation of the CDB:

a) At a point other than the boundary between two fields defined for the service action, or
b) At any point in the capability parameters (i.e., either all the capability parameters are included in the ADDITIONAL CDB LENGTH field or none of them are).

The SERVICE ACTION field indicates the action being requested by the application client. Each service action code description defines a number of service action specific fields that are needed for that service action.

5.1.2 Fields commonly used in service actions

OSD commands employ the basic structure shown in table 19 for the service action specific fields (see table 18) so that the same field is in the same location in all OSD CDBs. Fields that are unique to one or two CDBs are not shown in (see table 19).

Table 19 — OSD service action specific fields

Bit / Byte	7	6	5	4	3	2	1	0
10	OPTIONS BYTE							
11	SECOND OPTIONS BYTE							
12	(MSB)			OBJECT_GROUP_ID				
15								(LSB)
16	(MSB)			USER_OBJECT_ID				
23								(LSB)
24	(MSB)			OBJECT_SESSION_ID				
27								(LSB)
28	(MSB)			LENGTH				
35								(LSB)
36	(MSB)			STARTING BYTE ADDRESS				
43								(LSB)
44				Get attributes parameters				
55								
56				Set attributes parameters				
75								
76				Capabilities parameters				
167								

Editors Note 13 - ROW: The information in table 19 is for reference during the development of the OSD command formats. It will be removed as soon as the OSD command formats are stable.

5.1.2.1 Capabilities parameters

5.1.2.1.1 Capabilities parameters

The capabilities parameters (see table 20) are prepared and signed by a Policy/Storage Manager who gives them to an application or application client (see 4.4.5.4) to allow specified types of access to specified objects in the OSD over a specified period of time.

Table 20 — Capabilities parameters format

Bit / Byte	7	6	5	4	3	2	1	0
76	(MSB)							
91				CAPABILITIES DIGITAL SIGNATURE				(LSB)
92				Reserved				
93				Reserved				
94				MINIMUM SECURITY				
95				KEY IDENTIFIER		Reserved		WK_OBJ
96	(MSB)							
99				OBJECT_GROUP_ID				(LSB)
100	(MSB)							
115				SIGNATURE NONCE				(LSB)
116	(MSB)							
123				EXPIRATION TIME				(LSB)
124								
127				PERMISSIONS BIT MASK				
128								
130				Reserved				
131				Reserved		USER OBJECT SELECTOR		
132								
167				User object specific parameters				

The MINIMUM SECURITY field (see table 21) specifies which OSD service actions are allowed by the capabilities parameters.

Table 21 — Minimum security format

Bit / Byte	7	6	5	4	3	2	1	0
56	Reserved		M_IS_CDB	M_IS_DAT	Reserved		M_PS_CDB	M_PS_DAT

If the Must perform Integrity Security on the CDB and response data (M_IS_CDB) bit is zero, then no restrictions apply to the value of the IS_CDB bit in OSD commands associated with these capabilities parameters. If the M_IS_CDB bit is one and the IS_CDB bit is zero in an OSD CDB associated with these capabilities parameters, then the

command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

If the Must perform Integrity Security on Data transfers (M_IS_DAT) bit is zero, then no restrictions apply to the value of the IS_DATA bit in OSD commands associated with these capabilities parameters. If the M_IS_DAT bit is one and the IS_DATA bit is zero in an OSD CDB associated with these capabilities parameters, then the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

If the Must perform Privacy Security on the CDB and response data (M_PS_CDB) bit is zero, then no restrictions apply to the value of the PS_CDB bit in OSD commands associated with these capabilities parameters. If the M_PS_CDB bit is one and the PS_CDB bit is zero in an OSD CDB associated with these capabilities parameters, then the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

If the Must perform Privacy Security on Data transfers (M_PS_DAT) bit is zero, then no restrictions apply to the value of the PS_DATA bit in OSD commands associated with these capabilities parameters. If the M_PS_DAT bit is one and the PS_DATA bit is zero in an OSD CDB associated with these capabilities parameters, then the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

NOTE 3 The MINIMUM SECURITY field is structured so that its contents may be copied directly to the CDB of an OSD command with the result that the command meets the specified minimum security requirements.

The KEY IDENTIFIER field (see table 22) specifies which key in the security key hierarchy (see 4.4.5.2) applies to the capabilities digital signature in these capabilities parameters.

Table 22 — Key identifier values

Key Identifier	Key Name
0h	Master key
1h	Device key
2h	Object group key
3h	Object group working key A
4h	Object group working key B
5h - Fh	Reserved

A Well Known Object (WK_OBJ) bit of zero specifies that the capabilities parameters are restricted to user objects (see 4.4.2.5). A WK_OBJ bit of one specifies that the capability parameters allow operations on user objects and well known objects (see 4.4.2.2). If the OSD service action associated with these capabilities parameters attempts to act on a well known object and the WK_OBJ bit is zero, then the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

The OBJECT_GROUP_ID field specifies the object group to which the capabilities parameters apply. If the OBJECT_GROUP_ID field contains zero, the capabilities parameters apply to any object group. If the OSD service action associated with these capabilities parameters attempts to act on an object group other than the object group specified by the OBJECT_GROUP_ID field, then the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

The SIGNATURE NONCE field contains a random number generated by the Policy/Storage Manager. The random number should be generated using an algorithm that makes it difficult compute a new signature nonce value based on knowledge of a previous value. The random number should be generated in a way that reduces the likelihood of the same signature nonce value appearing two sets of capabilities parameters generated by a single or multiple OSD devices.

The EXPIRATION TIME field specifies the OSD time (see 4.4.5.5) after which these capabilities parameters are no longer valid. If the current OSD time is greater than the value in the EXPIRATION TIME field of the associated capabilities parameters, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

NOTE 4 OSD devices that experience a power cycle may invalid outstanding capabilities parameters by changing the keys in the key hierarchy associated with the KEY IDENTIFIER field.

The PERMISSIONS BIT MASK field (see table 23) specifies which OSD service actions are allowed by the capabilities parameters.

Table 23 — Permissions bit mask format

Bit Byte	7	6	5	4	3	2	1	0
86	RD	WRT	APP	OPNCLS	FLS_O	FLS_OG	CRE	CRE_OG
87	RMV	RMV_OG	GT_ATTR	ST_ATTR	LST	CRE_ATPG	RMV_ATPG	Reserved
88	Reserved							
89	Reserved							

Each bit in PERMISSIONS BIT MASK field (see table 24) identifies one OSD service action. If the bit is zero, the identified OSD service action is prohibited by the capabilities parameters. If the bit is one, the identified OSD service action is allowed.

Table 24 — Capabilities Permission Bits

Bit	Allowed Operation	Bit	Allowed Operation
APP	APPEND	OPNCLS	OPEN or CLOSE
CRE	CREATE or CREATE AND WRITE	RD	READ
CRE_ATPG	CREATE ATTRIBUTES PAGE	RMV	REMOVE
CRE_OG	CREATE OBJECT GROUP	RMV_ATPG	REMOVE ATTRIBUTES PAGE
FLS_O	FLUSH OBJECT	RMV_OG	REMOVE OBJECT GROUP
FLS_OG	FLUSH OBJECT GROUP	ST_ATTR	SET ATTRIBUTES
GT_ATTR	GET ATTRIBUTES	WRT	WRITE
LST	LIST		

If an OSD service action that is not allowed by PERMISSIONS BIT MASK is attempted, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

The USER OBJECT SELECTOR field (see table 25) specifies the user object selection criteria for these capability parameters and the format of the user object specific parameters.

Table 25 — user object selectors

Value	Selector		User Object Specific Parameters Reference
	Name	Description	
0h	NULL	There are no user object specific parameters in these capabilities parameters. These capabilities parameters apply to all otherwise qualified user objects. [a]	not used
1h	1USER OBJECT	The user object specific parameters select a single user object and a range of bytes within that user object to which these capability parameters apply.	5.1.2.1.2.1
2h	FS SPECIFIC	The user object specific parameters select zero or more user objects to which these capabilities apply using file system specific attributes.	5.1.2.1.2.2
3h - Fh	Reserved		

[a] The NULL user object selector is intended primarily for operations (e.g., CREATE OBJECT GROUP) that do not refer to a particular object.

The CAPABILITIES DIGITAL SIGNATURE field contains a digital signature (see 4.4.5.3) for all capabilities parameters fields except the CAPABILITIES DIGITAL SIGNATURE field. The device sever shall compute a digital signature for the capabilities parameters it receives and compare the computed value to the value found in the CAPABILITIES DIGITAL SIGNATURE field. If the computed value does not exactly match the value found in the CAPABILITIES DIGITAL SIGNATURE field, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

5.1.2.1.2 User object specific capabilities parameters

5.1.2.1.2.1 1USER OBJECT user object specific parameters

The 1USER OBJECT user object specific parameters select a single user object or a range of bytes within a single user object to which these capability parameters apply.

Table 26 — 1USER OBJECT user object specific parameters format

Bit / Byte	7	6	5	4	3	2	1	0
94	(MSB)			USER_OBJECT_ID				
101								(LSB)
102	(MSB)			LENGTH				
109								(LSB)
110	(MSB)			STARTING BYTE ADDRESS				
117								(LSB)
118	(MSB)			ACCESS CONTROL STATE				
119								(LSB)
120	(MSB)			USER OBJECT CREATION TIME				
127								(LSB)

The USER_OBJECT_ID field contain the User_Object_ID (see 4.4.2.5) of the selected user object. The command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB if:

a) The USER_OBJECT_ID field contains zero; or
b) The User_Object_ID of user object being acted upon by the OSD service action associated with these capabilities parameters does not match the contents of the USER_OBJECT_ID field.

The LENGTH and STARTING BYTE ADDRESS fields specify the range of bytes within a user object that may be acted upon by an OSD service actions associated with these capabilities parameters. If the LENGTH field contains zero, then the contents of the STARTING BYTE ADDRESS field shall be ignored and these capabilities parameters place no restrictions on the range of bytes within a user object that may be acted upon by an OSD service actions associated with these capabilities parameters.

If the contents of the LENGTH field are nonzero, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB if a READ, WRITE, or APPEND service action attempts to act upon a byte whose position in the user object is:

a) Less than the value in the STARTING BYTE ADDRESS field; or
b) Greater than the sum of the values in STARTING BYTE ADDRESS and LENGTH fields.

If the access control state in the user object attributes (see 5.2) of a user object accessed by an OSD service action does not exactly match the contents of the ACCESS CONTROL STATE field in these capabilities parameters, the

command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

Editors Note 14 - ROW: The wording of the preceding paragraph may change slightly with the structure of attributes is specified in detail.

The USER OBJECT CREATION TIME field allows selection of a user object based on the value of the creation time attribute in the User Object Timestamps attributes page (see 7.1.2.14). If the USER OBJECT CREATION TIME field contains zero, no comparison shall be made to the creation time user object attribute of the user object being accessed by an OSD service action. If the USER OBJECT CREATION TIME field is nonzero and the contents of the USER OBJECT CREATION TIME field do not match the creation time user object attribute, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

5.1.2.1.2.2 FS SPECIFIC user object specific parameters

Editors Note 15 - ROW: Conversion of this subclause has been delayed until the user object attributes are better understood.

The match flavor is defined to allow access to multiple objects satisfying predetermined properties. This may dramatically reduce the frequency of application clients obtaining new capabilities from Policy/Storage Managers. Because this flavor allows the capability creator to specify four separate values that may be found in an object's attributes, it subsumes the access control state and creation time restrictions of the single-range capability flavor. However, this flavor offers no analog to the range restrictions of the single-range capability flavor; it grants no access or access to any range of an object.

The flavor-specific field of the match flavor contains:

offset0 (8 bits)
offset1 (8 bits)
offset2 (8 bits)
offset3 (8 bits)
mask0 (32 bits)
mask1 (32 bits)
mask2 (32 bits)
mask3 (32 bits)
match0 (32 bits)
match1 (32 bits)
match2 (32 bits)
match3 (32 bits)

Definition: The accesses enumerated in the permissions word are permitted, provided that:

a) all requirements of the null flavor are met; and
b) for each N (N equals 0, 1, 2, and 3), the bitwise AND of maskN and the 4 bytes starting at byte offsetN of the FILE SYSTEM SPECIFIC field of the object's attributes is equal to matchN. This allows file managers to issue capabilities that are valid for objects satisfying certain properties that the file manager has pre-configured by setting specific values in the FILE SYSTEM SPECIFIC field of each object.

5.1.2.2 Get attributes parameters (pages only)

The get attributes CDB parameters defined in table 27 provide the ability for a service action to retrieve object attributes (see 4.4.3) in the form of two attributes pages.

Table 27 — Get attributes CDB parameters for data transfer operations

Bit / Byte	7	6	5	4	3	2	1	0
44	(MSB)							
47				FIRST GET ATTRIBUTES PAGE				(LSB)
48	(MSB)							
51				SECOND GET ATTRIBUTES PAGE				(LSB)
52	(MSB)							
55				GET ATTRIBUTES ALLOCATION LENGTH				(LSB)

The FIRST GET ATTRIBUTES PAGE and SECOND GET ATTRIBUTES PAGE fields specify one attribute page number each (see 7.1.2) to be retrieved. A zero in either field specifies that no attribute page is to be retrieved.

The GET ATTRIBUTES ALLOCATION LENGTH field specifies the number of bytes at the beginning of the Data-In Buffer allocated to receive retrieved attribute pages and attribute lists.

When the both the FIRST GET ATTRIBUTES PAGE and SECOND GET ATTRIBUTES PAGE fields are non-zero, the attribute page identified by the FIRST GET ATTRIBUTES PAGE field shall be placed in the Data-In Buffer starting at byte 0h and the other attributes page shall be placed in the Data-In Buffer starting at the first byte not used by the attribute page identified by the FIRST GET ATTRIBUTES PAGE field.

If the get attributes allocation length is not sufficient to accommodate all bytes in the specified attributes pages, the transfer of attributes data shall be truncated at the specified get attributes allocation length, this shall not be considered to be an error. When get attributes data is truncated, all the get attributes data that is transferred shall be transferred as if no error occurred (i.e., length fields in the transferred get attributes data shall not be modified to reflect the truncation).

5.1.2.3 Get attributes parameters (page and list)

The get attributes CDB parameters defined in table 28 provide the ability for a service action to retrieve object attributes (see 4.4.3) in the form of one attributes page and one attributes list.

Table 28 — Get attributes CDB parameters for operations that do not include data transfers

Bit / Byte	7	6	5	4	3	2	1	0
44	(MSB)							
47				GET ATTRIBUTES PAGE				(LSB)
48	(MSB)							
51				GET ATTRIBUTES LIST LENGTH				(LSB)
52	(MSB)							
55				GET ATTRIBUTES ALLOCATION LENGTH				(LSB)

The GET ATTRIBUTES PAGE field specifies an attribute page number (see 7.1.2) to be retrieved. A value of zero specifies that no attribute page is to be retrieved.

The GET ATTRIBUTES LIST LENGTH field specifies the length of a get attributes list (see 7.1.3) that specifies one or more attribute values to be retrieved. A get attributes list length of zero specifies that there is no get attributes list is included with the service action. The get attributes list shall be placed in the Data-Out buffer as described in 5.1.2.10.

The GET ATTRIBUTES ALLOCATION LENGTH field specifies the number of bytes at the beginning of the Data-In Buffer allocated to receive retrieved attribute pages and attribute lists.

When both the GET ATTRIBUTES PAGE and GET ATTRIBUTES LIST LENGTH fields are non-zero, the attribute page identified by the GET ATTRIBUTES PAGE field shall be placed in the Data-In Buffer starting at byte 0h and the list shall be placed in the Data-In Buffer starting at the first byte not used by the attribute page identified by the GET ATTRIBUTES PAGE field.

If the get attributes allocation length is not sufficient to accommodate all bytes in the specified attributes page and get attributes list, the transfer of attributes data shall be truncated at the specified get attributes allocation length, this shall not be considered to be an error. When get attributes data is truncated, all the get attributes data that is transferred shall be transferred as if no error occurred (i.e., length fields in the transferred get attributes data shall not be modified to reflect the truncation).

5.1.2.4 Length

The LENGTH field contains an unsigned 64 bit integer representing the number of bytes to be transferred by an OSD READ, WRITE, APPEND, or CREATE AND WRITE service action.

For the READ service action, the number of bytes in the Data-In Buffer shall be equal to the sum of the values in the LENGTH and GET ATTRIBUTES ALLOCATION LENGTH (see 5.1.2.2 and 5.1.2.3) fields. The byte in the Data-In Buffer where the first byte of read data is placed shall be identified by the contents of the GET ATTRIBUTES ALLOCATION LENGTH field (i.e., the get attributes data bytes encompass all Data-In Buffer bytes between byte 0h and the get attributes allocation length).

For the WRITE, APPEND, and CREATE AND WRITE service actions, the number of bytes in the Data-Out Buffer shall be equal to the sum of the values in the LENGTH and SET ATTRIBUTES PARAMETER LIST LENGTH (see 5.1.2.9) fields. The first byte of write data shall be located in the byte identified by the contents of the SET ATTRIBUTES PARAMETER LIST LENGTH field (i.e., the set attributes parameter data bytes encompass all Data-Out Buffer bytes between byte 0h and the set attributes parameter list length).

5.1.2.5 Object_Group_ID

The OBJECT_GROUP_ID field contains an Object_Group_ID (see 4.4.2.4) that the service action is to act upon.

5.1.2.6 Options Byte

Options Byte is a set of fields used to modify or control the service action (see table 29).

Table 29 — Option Byte format

Bit	7	6	5	4	3	2	1	0
	Reserved			DPO	FUA	Reserved		

The disable page out (DPO) bit allows the application client to influence the replacement of logical blocks in the cache. For write operations, setting this bit to one advises the device server to not replace existing blocks in the cache memory with the write data. For read operations, setting this bit to one causes blocks of data that are being read to not replace existing ones in the cache memory.

The force unit access (FUA) bit is used to indicate that the device server shall access the physical medium. For a write operation, setting FUA to one causes the device server to complete the data write to the physical medium before completing the command. For a read operation, setting FUA to one causes the logical blocks to be retrieved from the physical medium.

5.1.2.7 Object_Group_ID

The CDB OBJECT_GROUP_ID field contains the Object_Group_ID (see 4.4.2.4) that the service action is to act upon.

5.1.2.8 Object_Session_ID

The CDB OBJECT_SESSION_ID field specifies the Object_Session_ID (see 4.4.4.2) to be used during processing of the service action.

5.1.2.9 Set attributes parameters (values only)

The set attributes CDB parameters defined in table 30 provide the ability for a service action to set object attributes (see 4.4.3) in the form of two attributes values.

Table 30 — Set attributes values parameters

Bit / Byte	7	6	5	4	3	2	1	0
56	(MSB)							
59				FIRST SET ATTRIBUTES PAGE				(LSB)
60	(MSB)							
63				FIRST SET ATTRIBUTE NUMBER				(LSB)
64	(MSB)							
67				SECOND SET ATTRIBUTES PAGE				(LSB)
68	(MSB)							
71				SECOND SET ATTRIBUTE NUMBER				(LSB)
72	(MSB)							
75				SET ATTRIBUTES PARAMETER LIST LENGTH				(LSB)

The FIRST SET ATTRIBUTES PAGE and FIRST SET ATTRIBUTE NUMBER fields specify one attribute value to be set. A zero in the FIRST SET ATTRIBUTES PAGE field specifies that no attribute value is to be set. The length of an attribute shall not be altered when the attribute is set using FIRST SET ATTRIBUTES PAGE and FIRST SET ATTRIBUTE NUMBER fields.

If the FIRST SET ATTRIBUTES PAGE and FIRST SET ATTRIBUTE NUMBER fields specify an attribute that is unknown or has a length of zero, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

The SECOND SET ATTRIBUTES PAGE and SECOND SET ATTRIBUTE NUMBER fields specify a second attribute to be set. A zero in the SECOND SET ATTRIBUTES PAGE field specifies that no attribute value is to be set. The length of an attribute

shall not be altered when the attribute is set using SECOND SET ATTRIBUTES PAGE and SECOND SET ATTRIBUTE NUMBER fields.

If the SECOND SET ATTRIBUTES PAGE and SECOND SET ATTRIBUTE NUMBER fields specify an attribute that is unknown or has a length of zero, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

The SET ATTRIBUTES PARAMETER LIST LENGTH field specifies the number of bytes at the beginning of the Data-Out Buffer that contain parameter data to be used in the setting of attribute values.

If both the FIRST SET ATTRIBUTES PAGE and SECOND SET ATTRIBUTES PAGE fields are non-zero, the attribute value identified by the FIRST SET ATTRIBUTES PAGE and FIRST SET ATTRIBUTE NUMBER fields shall be placed in the Data-Out Buffer starting at byte 0h and the other attribute value shall be placed in the Data-Out Buffer starting at the first byte not used by the attribute value identified by the FIRST SET ATTRIBUTES PAGE and FIRST SET ATTRIBUTE NUMBER fields.

If the set attributes parameter list length causes the truncation of an attribute value, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

5.1.2.10 Set attributes parameters (value and list)

The set attributes CDB parameters defined in table 30 provide the ability for a service action to set object attributes (see 4.4.3) in the form of one attribute value and one attributes list.

Table 31 — Set attributes CDB parameters for operations that do not include data transfers

Bit / Byte	7	6	5	4	3	2	1	0
56	(MSB)							
59				SET ATTRIBUTES PAGE				(LSB)
60	(MSB)							
63				SET ATTRIBUTE NUMBER				(LSB)
64	(MSB)							
67				SET ATTRIBUTES LIST LENGTH				(LSB)
68	(MSB)							
71				Reserved				(LSB)
72	(MSB)							
75				SET ATTRIBUTES PARAMETER LIST LENGTH				(LSB)

The SET ATTRIBUTES PAGE and SET ATTRIBUTE NUMBER fields specify one attribute value to be set. A zero in the SET ATTRIBUTES PAGE field specifies that no attribute value is to be set. The length of an attribute shall not be altered when the attribute is set using SET ATTRIBUTES PAGE and SET ATTRIBUTE NUMBER fields.

If the SET ATTRIBUTES PAGE and SET ATTRIBUTE NUMBER fields specify an attribute that is unknown or has a length of zero, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

The SET ATTRIBUTES LIST LENGTH field specifies the length of a set attributes list (see 7.1.3) that specifies one or more attribute values to be set. A set attributes list length of zero specifies that there is no set attributes list included with the service action.

The SET ATTRIBUTES PARAMETER LIST LENGTH field specifies the number of bytes at the beginning of the Data-Out Buffer that contain parameter data to be used in the setting of attribute values.

If the GET ATTRIBUTES LIST LENGTH field (see 5.1.2.3) contains a non-zero value, the get attributes list shall be placed starting at byte 0h. If the SET ATTRIBUTES PAGE field is non-zero, the attribute value identified by the SET ATTRIBUTES PAGE and SET ATTRIBUTE NUMBER fields shall be placed in the Data-In Buffer starting at the first byte not used by the get attributes list. If SET ATTRIBUTES PARAMETER LIST LENGTH field is non-zero, set attributes the list shall be placed in the Data-In Buffer starting at the first byte not used by the attribute value identified by the SET ATTRIBUTES PAGE and SET ATTRIBUTE NUMBER fields.

If the set attributes parameter list length causes the truncation of the attribute value or entry in the attribute list, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

5.1.2.11 Starting byte address

The STARTING BYTE ADDRESS field specifies the location where the read or write is to commence in the specified object relative to the first byte (i.e., byte zero) of the object.

5.1.2.12 User_Object_ID

The USER_OBJECT_ID field contains the User_Object_ID of the user object (see 4.4.2.5) that the service action is to act upon.

5.2 Data Storage Policies

Policy refers to the set of conditions and subsequent actions (to be performed in the event the associated condition(s) is(are) met) connected to the management of storage. Policies are typically time or event driven, are independent of storage geometry, and frequently occur independently of any application processes. Common examples of policies include conditional or time-based backup, archive, delete processing, data movement, and device maintenance.

Policies, though relevant to the management and disposition of objects and their contents, are beyond the scope of this standard.

Editors Note 16 - GEM: Should we leave policies beyond the scope, add within the scope a method to convey policies, or delete them altogether?

5.2.1 Setting object session parameter values

5.2.1.1 General structure

The general command modifiers for setting parameters can be viewed as:

Table 32 — Parameter Modifiers

Parameter Identifier	Comparator	Low/Only Value	High Value

Where:

a) Parameter Identifier - indicates the particular parameter being set. The identifier enables discriminating among static, dynamic, and extended parameters;
b) Comparator - is one of 'value', 'less than', 'greater than', or 'inclusive'. value indicates a specific desired amount (specified by Low/Only Value below) is given. less than and greater than indicate that the Low/Only Value is to be viewed as the maximum or minimum values (respectively) for the parameter. inclusive indicates that the parameter is to lie within the range specified by Low/Only Value and High Value (see below);
c) Low/Only Value - an integer representing the bottom of a range, if one is indicated by the comparator; otherwise, it is the unique value; and
d) High Value - an integer representing the top of a range, if one is indicated by the comparator; otherwise it is absent.

Editors Note 17 - GEM: Should absent be changed to zero?

Attribute retrieval commands only utilize the attribute identifier.

6 Commands for OSD devices

6.1 Summary of commands for OSD devices

The commands in table 33 are the operations that an OSD device may perform as its contribution to the OSA environment. The SERVICE ACTION field in the CDB uniquely identifies the operation to take place. The command subclauses describe the service provided by that operation and the information that shall be passed to the OSD device in order for it to perform that function. The information that could be returned by the storage device to the application client is also listed.

Table 33 — Commands for OSD devices (part 1 of 2)

Command name	Operation code	Service action [a]	Type	Reference
APPEND	7Fh	8807h	M	6.2
CHANGE ALIASES	A4h	0Bh	O	SPC-3
CLOSE	7Fh	8809h	O	6.3
CREATE	7Fh	8802h	M	6.4
CREATE AND WRITE	7Fh	8812h	M	6.5
CREATE ATTRIBUTES PAGE	7Fh	8813h	O	6.6
CREATE OBJECT GROUP	7Fh	880Bh	M	6.7
FORMAT OSD	7Fh	8801h	M	6.8
FLUSH OBJECT	7Fh	8808h	M	6.9
GET ATTRIBUTES	7Fh	880Eh	M	6.10
IMPORT USER OBJECT	7Fh	880Dh	O	6.11
INQUIRY	12h		M	SPC-3
LIST	7Fh	8803h	M	6.12
LOG SELECT	4Ch		O	SPC-3
LOG SENSE	4Dh		O	SPC-3
MODE SELECT(10)	55h		O	SPC-3
MODE SENSE(10)	5Ah		O	SPC-3
OPEN	7Fh	8804h	O	6.13
OSD SYNCHRONIZE CACHE	7Fh	8811h	O	
PERSISTENT RESERVE IN	5Eh		M	SPC-3
PERSISTENT RESERVE OUT	5Fh		M	SPC-3
PREVENT ALLOW MEDIUM REMOVAL	1Eh		O	SPC-3
READ	7Fh	8805h	M	6.14
READ BUFFER	3Ch		O	SPC-3
RECEIVE COPY RESULTS	84h		O	SPC-3
RECEIVE DIAGNOSTIC RESULTS	1Ch		O	SPC-3
REMOVE	7Fh	880Ah	M	6.15
REMOVE ATTRIBUTE PAGE	7Fh	8814h	O	6.16

Key: M = Command implementation is mandatory.
O = Command implementation is optional.
OB = Command implementation is defined in a previous standard

[a] No entry in the service action column means that the SERVICE ACTION field does not apply to the command. OSD service action codes not listed in this table (i.e., 8800h and 8810h-88FFh) are reserved for future standardization

Table 33 — Commands for OSD devices (part 2 of 2)

Command name	Operation code	Service action [a]	Type	Reference
REMOVE OBJECT GROUP	7Fh	880Ch	M	6.17
REPORT ALIASES	A3h	0Bh	O	SPC-3
REPORT LUNS	A0h		O	SPC-3
REQUEST SENSE	03h		M	SPC-3
SEND DIAGNOSTIC	1Dh		M	SPC-3
SET ATTRIBUTES	7Fh	880Fh	M	6.18
START STOP UNIT	1Bh		O	SBC-2
TEST UNIT READY	00h		M	SPC-3
WRITE	7Fh	8806h	M	6.19
WRITE BUFFER	3Bh		Z	SPC-3

Key: M = Command implementation is mandatory.
O = Command implementation is optional.
OB = Command implementation is defined in a previous standard

[a] No entry in the service action column means that the SERVICE ACTION field does not apply to the command. OSD service action codes not listed in this table (i.e., 8800h and 8810h-88FFh) are reserved for future standardization

6.2 APPEND

The APPEND service action (see table 34) shall cause the specified number of bytes to be written to the designated object starting immediately after the object logical length.

Table 34 — APPEND service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8807h)				(LSB)
10				OPTIONS BYTE				
11				Reserved		PRIORITY		
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				USER_OBJECT_ID				(LSB)
24	(MSB)							
27				OBJECT_SESSION_ID				(LSB)
28	(MSB)							
35				LENGTH				(LSB)
36								
43				Reserved				
44								
55				Get attributes parameters (pages only)				
56								
75				Set attributes parameters (values only)				
76								
167				Capabilities parameters				

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The PRIORITY field establishes the priority of this command relative to other commands being processed by the same logical unit. Commands other than READ, WRITE, and APPEND commands have a priority of 7h. Priority 0h is the highest priority, with increasing PRIORITY values indicating lower priorities.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The contents of the OBJECT_SESSION_ID field are defined in 5.1.2.8.

The contents of the LENGTH field are defined in 5.1.2.4. The data to be written to the user object shall be placed in the Data-Out Buffer as described in 5.1.2.4.

If there is a capacity quota on the object group for this object, the OSD device shall test to make sure the APPEND does not exceed the quota. If quota is exceeded, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

The get attributes parameters are defined in 5.1.2.2. The Object Write attributes page (see 7.1.2.16) returns useful information for the APPEND service action.

The set attributes parameters are defined in 5.1.2.9.

The capabilities parameters are defined in 5.1.2.1. If the OBJECT_SESSION_ID field contains a nonzero value, the capabilities parameters may be omitted. If the capabilities parameters are omitted, the device server shall use the capabilities parameters saved in association with the object session.

6.3 CLOSE

The CLOSE service action (see table 35) shall cause the specified user object to be identified as no longer in use by a given object session.

Table 35 — CLOSE service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8809h)				(LSB)
10				OPTIONS BYTE				
11				Reserved				
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				USER_OBJECT_ID				(LSB)
24	(MSB)							
27				OBJECT_SESSION_ID				(LSB)
28								
43				Reserved				
44								
55				Get attributes parameters (page and list)				
56								
75				Set attributes parameters (value and list)				
76								
167				Capabilities parameters				

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The contents of the OBJECT_SESSION_ID field are defined in 5.1.2.8.

The get attributes parameters are defined in 5.1.2.3.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.4 CREATE

The CREATE service action (see table 36) causes the OSD device to allocate and initialize a user object.

Table 36 — CREATE service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8802h)				(LSB)
10				OPTIONS BYTE				
11				Reserved			SESSION	Reserved
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16								
43				Reserved				
44								
55				Get attributes parameters (page and list)				
56								
75				Set attributes parameters (value and list)				
76								
167				Capabilities parameters				

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

A SESSION bit of zero specifies that the CREATE service action shall not open a object session for the newly created object. A SESSION bit of one specifies that the CREATE service action shall open a object session for the newly created object.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The get attributes parameters are defined in 5.1.2.3. The Session attributes page (see 7.1.2.18) returns useful information for the CREATE service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.5 CREATE AND WRITE

The CREATE AND WRITE service action (see table 36) causes the OSD device to allocate and initialize a user object and then write data to the newly created user object. The CREATE AND WRITE service action is identical to the CREATE service action (see 6.4) except that only attribute pages may be retrieved and only attributes values may be set.

Table 37 — CREATE AND WRITE service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8812h)				(LSB)
10				OPTIONS BYTE				
11				Reserved			SESSION	Reserved
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16								
27				Reserved				
28	(MSB)							
35				LENGTH				(LSB)
36	(MSB)							
43				STARTING BYTE ADDRESS				(LSB)
44								
55				Get attributes parameters (pages only)				
56								
75				Set attributes parameters (values only)				
76								
145				Capabilities parameters				

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

A SESSION bit of zero specifies that the CREATE AND WRITE service action shall not open a object session for the newly created object. A SESSION bit of one specifies that the CREATE AND WRITE service action shall open a object session for the newly created object.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the LENGTH field are defined in 5.1.2.4. The data to be written to the user object shall be placed in the Data-Out Buffer as described in 5.1.2.4.

The contents of the STARTING BYTE ADDRESS field are defined in 5.1.2.11.

The get attributes parameters are defined in 5.1.2.2. The Session attributes page (see 7.1.2.18) returns useful information for the object creation function of the CREATE AND WRITE service action. The Object Write attributes page (see 7.1.2.16) returns useful information for the write function of the CREATE AND WRITE service action.

The set attributes parameters are defined in 5.1.2.9.

The capabilities parameters are defined in 5.1.2.1.

6.6 CREATE ATTRIBUTES PAGE

The CREATE ATTRIBUTES PAGE service action (see table 38) initializes a new attributes page (see 4.4.3.2) or replaces a previously created attributes page that is associated with the specified object.

Table 38 — CREATE ATTRIBUTES PAGE service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8813h)				(LSB)
10				Reserved				
11				Reserved				REF_PG
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				USER_OBJECT_ID				(LSB)
24								
27				Reserved				
28	(MSB)							
31				ATTRIBUTE PAGE RANGE BEGIN				(LSB)
32	(MSB)							
35				ATTRIBUTE PAGE RANGE END				(LSB)
36	(MSB)							
39				PARAMETER LIST LENGTH				(LSB)
40								
43				Reserved				
44								
55				Get attributes parameters (pages only)				
56								
75				Reserved				
76								
167				Capabilities parameters				

The reference page (REF_PG) bit specifies whether the attributes page to be created is a page of references to other attributes pages. If the REF_PG bit is set to zero, the attributes pages to be created contains attributes values. If the REF_PG bit is set to one, the attributes pages to be created contains references to attributes values located in other attributes pages.

Support for dynamic creation of attributes pages containing references to other attributes pages is optional. If the REF_PG bit is set to one and the dynamic creation of attributes pages containing references to other attributes pages is not supported, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The contents of the OBJECT_GROUP_ID field (see 5.1.2.7) specify the Object_Group_ID of the object with which the attributes page is associated.

The contents of the USER_OBJECT_ID field (see 5.1.2.12) specify the User_Object_ID of the object with which the attributes page is associated.

The ATTRIBUTE PAGE RANGE BEGIN field specifies the lowest valued attribute page number from which the device server shall assign an attribute page number to the newly created attributes page.

The ATTRIBUTE PAGE RANGE END field specifies the highest valued attribute page number from which the device server shall assign an attribute page number to the newly created attributes page.

If both the ATTRIBUTE PAGE RANGE BEGIN field and the ATTRIBUTE PAGE RANGE END field contain zero, the device server shall assign an attribute page number from the set of dynamically assigned page numbers that conforms to the attribute page numbering rules described in 4.4.3.2.

If both the ATTRIBUTE PAGE RANGE BEGIN field and the ATTRIBUTE PAGE RANGE END field contain the same value and that value is the attributes page number of a previously created attributes page, the contents of the attributes page shall be replaced.

If the device server is unable to assign an attribute page number from the range specified by the ATTRIBUTE PAGE RANGE BEGIN and ATTRIBUTE PAGE RANGE END fields that conforms to the attribute page numbering rules described in 4.4.3.2, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The PARAMETER LIST LENGTH field specifies the number of bytes of parameter data for the CREATE ATTRIBUTES PAGE service action.

The get attributes parameters are defined in 5.1.2.2. The attribute page number assigned by the CREATE ATTRIBUTES PAGE service action may be obtained from the Current Command attributes page (see 7.1.2.15).

The capabilities parameters are defined in 5.1.2.1.

The format of the CREATE ATTRIBUTE PAGE service action parameter list is shown in table 39.

Table 39 — CREATE ATTRIBUTE PAGE parameter list format

Bit / Byte	7	6	5	4	3	2	1	0
0	ATTRIBUTE 0 VALUE							
39								
40	Attributes list							
n								

The ATTRIBUTE 0 VALUE field specifies the value to be stored in attribute 0 (see 7.1.2.2) of the newly created attributes page.

The attributes list specifies values, other than the attribute number 0h value, to be stored in the newly created attributes page in attributes list format (see 7.1.3). If the REF_PG bit is set to zero, the LIST TYPE field in the attributes list shall contain 9h. If the REF_PG bit is set to one, the LIST TYPE field in the attributes list shall contain 1h.

6.7 CREATE OBJECT GROUP

The CREATE OBJECT GROUP service action (see table 40) shall cause the OSD device to allocate a new object group and establish an new group object for the object group.

Table 40 — CREATE OBJECT GROUP service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (880Bh)				(LSB)
10				OPTIONS BYTE				
11								
27				Reserved				
28	(MSB)							
35				CAPACITY QUOTA				(LSB)
36								
43				Reserved				
44								
55				Get attributes parameters (page and list)				
56								
75				Set attributes parameters (value and list)				
76								
167				Capabilities parameters				

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The CAPACITY QUOTA field specifies the limit on the number of bytes that may be allocated by the newly created object group. If the CAPACITY QUOTA field contains zero, the number of bytes that may be allocated by the object group is unlimited.

The get attributes parameters are defined in 5.1.2.3. The Object Group attributes page (see 7.1.2.17) returns useful information for the CREATE OBJECT GROUP service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.8 FORMAT OSD

The FORMAT OSD service action (see table 41) causes the OSD device to delete all user objects, delete all group objects, and set the attributes for the root object to defaults. There are no parameters or other identifiers required in this service action. It results in attribute structures being constructed that support the creation and access of objects.

Table 41 — FORMAT OSD service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (8801h)				
9								(LSB)
10	OPTIONS BYTE							
11	Reserved							
27								
28	(MSB)			FORMATTED CAPACITY				
35								(LSB)
36	Reserved							
43								
44	Get attributes parameters (page and list)							
55								
56	Set attributes parameters (value and list)							
75								
76	Capabilities parameters							
167								

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The FORMATTED CAPACITY field specifies the total capacity of the formatted OSD logical unit in bytes. If the FORMATTED CAPACITY field is set to zero, the entire OSD device is formatted as an OSD device and the logical unit capacity established accordingly. If value in the FORMATTED CAPACITY field is greater than the maximum OSD device capacity, the formatting operation shall process as if the value were zero, this shall not be considered an error.

The get attributes parameters are defined in 5.1.2.3. The Logical Unit attributes page (see 7.1.2.19) returns useful information for the FORMAT OSD service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.9 FLUSH OBJECT

The FLUSH OBJECT service action (see table 42) ensures all data and attribute bytes for the specified object are stored in non-volatile media.

Table 42 — FLUSH OBJECT service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (8808h)				
9								(LSB)
10	Reserved							
11	Reserved							
12	(MSB)			OBJECT_GROUP_ID				
15								(LSB)
16	(MSB)			USER_OBJECT_ID				
23								(LSB)
24	Reserved							
43								
44	Get attributes parameters (page and list)							
55								
56	Set attributes parameters (value and list)							
75								
76	Capabilities parameters							
167								

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The get attributes parameters are defined in 5.1.2.3.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.10 GET ATTRIBUTES

The GET ATTRIBUTES service action (see table 43) instructs the device server to return the attributes.

Table 43 — GET ATTRIBUTES service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (880Eh)				(LSB)
10				Reserved				
11				Reserved				
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				USER_OBJECT_ID				(LSB)
24	(MSB)							
27				OBJECT_SESSION_ID				(LSB)
28								
43				Reserved				
44								
55				Get attributes parameters (page and list)				
56								
75				Set attributes parameters (value and list)				
76								
167				Capabilities parameters				

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The contents of the OBJECT_SESSION_ID field are defined in 5.1.2.8.

The get attributes parameters are defined in 5.1.2.3.

Security keys shall not be returned by the GET ATTRIBUTES service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.11 IMPORT USER OBJECT

The IMPORT USER OBJECT service action (see table 44) causes the OSD device to access another OSD device, retrieve a specified user object, and create copy of the user object on the OSD that receives the IMPORT USER OBJECT service action. This service action shall copy a user object from another OSD device by issuing OPEN, READs and a CLOSE or just a READ to the designated OSD device.

Table 44 — IMPORT USER OBJECT service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (880Dh)				(LSB)
10				OPTIONS BYTE				
11				Reserved				
12	(MSB)							
15				SOURCE OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				SOURCE USER_OBJECT_ID				(LSB)
24	(MSB)							
27				DESTINATION OBJECT_GROUP_ID				(LSB)
28	(MSB)							
35				SOURCE OSD DEVICE ALIAS VALUE				(LSB)
36	(MSB)							
43				SOURCE OSD LOGICAL UNIT NUMBER				(LSB)
44								
55				Get attributes parameters (page and list)				
56								
75				Set attributes parameters (value and list)				
76								
167				Capabilities parameters				

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The SOURCE OBJECT_GROUP_ID and SOURCE USER_OBJECT_ID fields specify the Object_Group_ID (see 4.4.2.4) and User_Object_ID (see 4.4.2.5) of the user object that is the source for the copy operation.

The DESTINATION OBJECT_GROUP_ID field specifies the Object_Group_ID of the object group on the OSD device that receives the IMPORT USER OBJECT service action in which a new user object is to be created to be the destination for the copy operation.

The SOURCE OSD DEVICE ALIAS VALUE field specifies an alias value that identifies the SCSI target device containing the source OSD device. Alias values are established using the CHANGE ALIASES command (see SPC-3).

The SOURCE OSD LOGICAL UNIT NUMBER field specifies logical unit number of the OSD device within the SCSI target device specified by the SOURCE OSD DEVICE ALIAS VALUE field.

The get attributes parameters are defined in 5.1.2.3. The Session attributes page (see 7.1.2.18) returns useful information for the IMPORT USER OBJECT service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.12 LIST

The LIST service action is used to get information from the root object or a group object.

Table 45 — LIST service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8803h)				(LSB)
10								
11				Reserved				
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16								
18				Reserved				
19	INDEX				SORT ORDER			
20								
27				Reserved				
28	(MSB)							
31				ALLOCATION LENGTH				(LSB)
32								
43				Reserved				
44								
55				Get attributes parameters (pages only)				
56								
75				Set attributes parameters (values only)				
76								
167				Capabilities parameters				

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7. If the OBJECT_GROUP_ID field contains zero, the Object_Group_IDs in the root object shall be returned. If the OBJECT_GROUP_ID field contains a nonzero value, the User_Object_IDs in the specified object group shall be returned.

The INDEX field specifies the starting position of the IDs within the specified sort order, with initial position value of zero.

Editors Note 18 - ROW: Anybody know what this means? If no answers are forthcoming, the index field will be removed from a future revision.

The SORT ORDER field (see table 46) specifies the order in which the returned Object_Group_IDs or User_Object_IDs shall be sorted.

Table 46 — LIST sort order values

Sort Order	Description
0h	Vendor specific
1h	Ascending numeric value
2h	Descending numeric value
3h - Fh	Reserved

The ALLOCATION LENGTH field specifies the maximum number of bytes that an application client has allocated for returned list. An allocation length of zero indicates that no data shall be transferred. This condition shall not be considered as an error.

The allocation length is used to limit the maximum amount of the list returned to an application client. The device server shall terminate transfers to the Data-In Buffer when allocation length bytes have been transferred or when all available data have been transferred, whichever is less. If the information being transferred is truncated, the contents of the ADDITIONAL LENGTH field (see table 47) shall not be altered to reflect the truncation.

If the amount of information to be transferred exceeds the maximum value that may be specified in the ALLOCATION LENGTH field the device server shall transfer no data and return a CHECK CONDITION status; the sense key shall be set to ILLEGAL REQUEST and the additional sense code shall be set to INVALID FIELD IN CDB.

The number of bytes in the Data-In Buffer shall be equal to the sum of the values in the LENGTH and GET ATTRIBUTES ALLOCATION LENGTH (see 5.1.2.2 and 5.1.2.3) fields. The byte in the Data-In Buffer where the first byte of the list is placed shall be identified by the contents of the GET ATTRIBUTES ALLOCATION LENGTH field (i.e., the get attributes data bytes encompass all Data-In Buffer bytes between byte 0h and the get attributes allocation length).

The get attributes parameters are defined in 5.1.2.2

The set attributes parameters are defined in 5.1.2.9.

The capabilities parameters are defined in 5.1.2.1.

The parameter data returned by the LIST service action (see table 47) contains information about the specified object groups or user objects.

Table 47 — LIST service action parameter data

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3	ADDITIONAL LENGTH (n-3)							(LSB)
4	Reserved							
6								
7	Reserved							ROOT
8	List of User_Object_IDs or Object_Group_IDs							
n								

The ADDITIONAL LENGTH field indicates the number of bytes of LIST service action parameter data that follow. If the parameter data is truncated due to insufficient allocation length, the ADDITIONAL LENGTH field shall not be altered to reflect the truncation.

A ROOT bit of zero indicates that the object IDs in the parameter data are from an object group and are 8-byte User_Object_IDs. A ROOT bit of one indicates that the object IDs in the parameter data are from the root object and are 4-byte Object_Group_IDs.

The list of User_Object_IDs or Object_Group_IDs contains one entry for each user object or object group identified by the LIST service action.

6.13 OPEN

The OPEN service action (see table 48) communicates to the OSD device that a user object (see 4.4.2.5) is to be accessed. The OPEN service action allows the OSD device to prefetch read data for the specified user object. The application client may optionally start an object session via the OPEN and specify the capabilities to be used by all READ, WRITE and APPEND services actions processed as part of the object session.

Support for the OPEN service action is optional. If the OPEN service action is supported, the CLOSE service action (see 6.3) shall also be supported.

Table 48 — OPEN service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (8804h)				
9								(LSB)
10	OPTIONS BYTE							
11	Reserved							SESSION
12	(MSB)			OBJECT_GROUP_ID				
15								(LSB)
16	(MSB)			USER_OBJECT_ID				
23								(LSB)
24				Reserved				
27								
28	(MSB)			PREFETCH LENGTH				
35								(LSB)
36				Reserved				
43								
44				Get attributes parameters (page and list)				
55								
56				Set attributes parameters (value and list)				
75								
76				Capabilities parameters				
167								

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

A SESSION bit of zero specifies that no object session is to be started for the user object. A SESSION bit of one specifies that the OSD device shall start an object session for the user object.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The PREFETCH LENGTH field specifies the minimum number of bytes of data that should be prefeteched from the user object as part of OPEN processing. The PREFETCH LENGTH field may be zero, this shall not be considered an error.

The get attributes parameters are defined in 5.1.2.3. The Session attributes page (see 7.1.2.18) returns useful information for the OPEN service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1. If the SESSION bit is one, the capabilities parameters from the OPEN shall be saved with other information about the newly created object session. READ, WRITE, and APPEND service actions that specify the assigned Object_Session_ID and do not include capabilities parameters in their CDBs shall have the saved capabilities parameters from the object session OPEN applied to them.

6.14 READ

The READ service action (see table 49) requests that the device server to return data to the application client from a specified user object.

Table 49 — READ service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (8805h)				
9								(LSB)
10	OPTIONS BYTE							
11	Reserved				PRIORITY			
12	(MSB)			OBJECT_GROUP_ID				
15								(LSB)
16	(MSB)			USER_OBJECT_ID				
23								(LSB)
24	(MSB)			OBJECT_SESSION_ID				
27								(LSB)
28	(MSB)			LENGTH				
35								(LSB)
36	(MSB)			STARTING BYTE ADDRESS				
43								(LSB)
44				Get attributes parameters (pages only)				
55								
56				Set attributes parameters (values only)				
75								
76				Capabilities parameters				
167								

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The PRIORITY field establishes the priority of this command relative to other commands being processed by the same logical unit. Commands other than READ, WRITE, and APPEND commands have a priority of 7h. Priority 0h is the highest priority, with increasing PRIORITY values indicating lower priorities.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The contents of the OBJECT_SESSION_ID field are defined in 5.1.2.8.

The contents of the LENGTH field are defined in 5.1.2.4. The data read from the user object shall be placed in the Data-In Buffer as described in 5.1.2.4.

The contents of the STARTING BYTE ADDRESS field are defined in 5.1.2.11.

If the values in the LENGTH and STARTING BYTE ADDRESS fields result an attempt to read a byte that is beyond the object logical length in the User Object Resources attributes page (see 7.1.2.11), no bytes shall be transferred and the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

The get attributes parameters are defined in 5.1.2.2. The Session attributes page (see 7.1.2.18) returns useful information for the READ service action.

The set attributes parameters are defined in 5.1.2.9.

The capabilities parameters are defined in 5.1.2.1. If the OBJECT_SESSION_ID field contains a nonzero value, the capabilities parameters may be omitted. If the capabilities parameters are omitted, the device server shall use the capabilities parameters saved in association with the object session.

6.15 REMOVE

The REMOVE service action (see table 50) deletes a user object.

Table 50 — REMOVE service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (880Ah)				
9								(LSB)
10	OPTIONS BYTE							
11	Reserved							
12	(MSB)			OBJECT_GROUP_ID				
15								(LSB)
16	(MSB)			USER_OBJECT_ID				
23								(LSB)
24				Reserved				
43								
44				Get attributes parameters (page and list)				
55								
56				Set attributes parameters (value and list)				
75								
76				Capabilities parameters				
167								

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The get attributes parameters are defined in 5.1.2.3. The Object Group attributes page (see 7.1.2.17) returns useful information for the REMOVE service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.16 REMOVE ATTRIBUTES PAGE

The REMOVE ATTRIBUTES PAGE service action (see table 51) removes an attributes page (see 4.4.3.2) associated with the specified object.

Table 51 — REMOVE ATTRIBUTES PAGE service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8814h)				(LSB)
10				Reserved				
11				Reserved				
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				USER_OBJECT_ID				(LSB)
24								
27				Reserved				
28	(MSB)							
31				ATTRIBUTES PAGE				(LSB)
32								
43				Reserved				
44								
55				Get attributes parameters (page and list)				
56								
75				Set attributes parameters (value and list)				
76								
167				Capabilities parameters				

The contents of the OBJECT_GROUP_ID field (see 5.1.2.7) specify the Object_Group_ID of the object with which the attributes page to be removed is associated.

The contents of the USER_OBJECT_ID field (see 5.1.2.12) specify the User_Object_ID of the object with which the attributes page to be removed is associated.

The ATTRIBUTES PAGE field the number of the attributes page to be removed.

The get attributes parameters are defined in 5.1.2.3.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

6.17 REMOVE OBJECT GROUP

The REMOVE OBJECT GROUP service action shall delete an object group from the OSD device. Any user objects the object group contains shall be deleted.

Table 52 — REMOVE OBJECT GROUP service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (880Ch)				
9								(LSB)
10	OPTIONS BYTE							
11	Reserved							
12	(MSB)			OBJECT_GROUP_ID				
15								(LSB)
16	Reserved							
43								
44	Get attributes parameters (page and list)							
55								
56	Set attributes parameters (value and list)							
75								
76	Capabilities parameters							
167								

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The get attributes parameters are defined in 5.1.2.3. The Logical Unit attributes page (see 7.1.2.19) returns useful information for the REMOVE OBJECT GROUP service action.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

Editors Note 19 - GEM: What is the outcome if a user object is a member of more than one object group? ROW: What causes that condition to come into existence?

6.18 SET ATTRIBUTES

The SET ATTRIBUTES service action (see table 53) shall set attributes for a specified root object, object group, or user object.

Table 53 — SET ATTRIBUTES service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (880Eh)				(LSB)
10				Reserved				
11				Reserved				
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				USER_OBJECT_ID				(LSB)
24	(MSB)							
27				OBJECT_SESSION_ID				(LSB)
28								
43				Reserved				
44								
55				Get attributes parameters (page and list)				
56								
75				Set attributes parameters (value and list)				
76								
167				Capabilities parameters				

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The contents of the OBJECT_SESSION_ID field are defined in 5.1.2.8. If the OBJECT_SESSION_ID field contains a nonzero value, the contents of the OBJECT_GROUP_ID and USER_OBJECT_ID fields shall be ignored and the attributes shall be set only for the specified object session.

The get attributes parameters are defined in 5.1.2.3.

The set attributes parameters are defined in 5.1.2.10.

The capabilities parameters are defined in 5.1.2.1.

Security keys (see 5.3.2.1) are the only attributes able to be set via the SET ATTRIBUTES service action that the GET ATTRIBUTES service action is prohibited from returning.

6.19 WRITE

The WRITE service action (see table 54) shall cause the specified number of bytes to be written to the designated object at the relative location specified.

Table 54 — WRITE service action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9				SERVICE ACTION (8806h)				(LSB)
10	OPTIONS BYTE							
11	Reserved				PRIORITY			
12	(MSB)							
15				OBJECT_GROUP_ID				(LSB)
16	(MSB)							
23				USER_OBJECT_ID				(LSB)
24	(MSB)							
27				OBJECT_SESSION_ID				(LSB)
28	(MSB)							
35				LENGTH				(LSB)
36	(MSB)							
43				STARTING BYTE ADDRESS				(LSB)
44								
55				Get attributes parameters (pages only)				
56								
75				Set attributes parameters (values only)				
76								
167				Capabilities parameters				

The contents of the OPTIONS BYTE field are defined in 5.1.2.6.

The PRIORITY field establishes the priority of this command relative to other commands being processed by the same logical unit. Commands other than READ, WRITE, and APPEND commands have a priority of 7h. Priority 0h is the highest priority, with increasing PRIORITY values indicating lower priorities.

The contents of the OBJECT_GROUP_ID field are defined in 5.1.2.7.

The contents of the USER_OBJECT_ID field are defined in 5.1.2.12.

The contents of the OBJECT_SESSION_ID field are defined in 5.1.2.8.

The contents of the LENGTH field are defined in 5.1.2.4. The data to be written to the user object shall be placed in the Data-Out Buffer as described in 5.1.2.4.

The contents of the STARTING BYTE ADDRESS field are defined in 5.1.2.11.

A WRITE to a byte that is greater than the object logical length in the User Object Resources attributes page (see 7.1.2.11) shall implicitly increase the logical length of the object. If there is a capacity quota on the object group for this object, the OSD device shall test to make sure the WRITE does not exceed the quota. If quota is exceeded, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

The get attributes parameters are defined in 5.1.2.2. The Object Write attributes page (see 7.1.2.16) returns useful information for the WRITE service action.

The set attributes parameters are defined in 5.1.2.9.

The capabilities parameters are defined in 5.1.2.1. If the OBJECT_SESSION_ID field contains a nonzero value, the capabilities parameters may be omitted. If the capabilities parameters are omitted, the device server shall use the capabilities parameters saved in association with the object session.

7 Parameters for OSD type devices

7.1 Attributes parameters

7.1.1 Attribute parameter formats

The following formats shall be provided for attribute parameter data:

a) Page format (see 7.1.2); and
b) List format (see 7.1.3).

Page format parameter data groups attributes in collections of related attribute values and provides access to them in formatted pages where only the attribute values appear in the parameter data. In page format, attribute access is limited to the attributes associated with the user object specified by a service action, the object group of which that user object is a member, and the root object. Those attribute pages that do not have a defined page format are inaccessible via page format parameter data (e.g., the Root Directory attributes page defined in 7.1.2.3).

List format parameter data handles individual attributes in an identifier, length, value format, using attribute pages as part of a multi-level identifier mechanism. Using the list format, any attribute known to the OSD logical unit is accessible.

7.1.2 OSD attribute pages

7.1.2.1 Attribute pages overview

Every attribute page is identified by a page number (see 4.4.3.2). Every attribute page includes attribute number 0h whose contents are defined in 7.1.2.2.

In addition to attribute number 0h, an attribute page is composed of:

a) Attribute values; or
b) References to attributes in other attribute pages.

If one entry in an attribute page is a reference to an attribute in another attribute page, all entries in that attribute page shall be references to other attribute values.

The attribute pages defined by this standard are shown in table 55.

Table 55 — OSD Attribute Pages

Page Number	Page Name	Reference
0h	User Object Directory	7.1.2.5
1h	User Object Information	7.1.2.8
2h	User Object Resources	7.1.2.11
3h	User Object Timestamps	7.1.2.14
4h - 7Fh	Reserved	
G+0h	Group Directory	7.1.2.4
G+1h	Group Information	7.1.2.7
G+2h	Group Resources	7.1.2.10
G+3h	Group Timestamps	7.1.2.13
G+4h - G+7Fh	Reserved	
R+0h	Root Directory	7.1.2.3
R+1h	Root Information	7.1.2.6
R+2h	Root Resources	7.1.2.9
R+3h	Root Timestamps	7.1.2.12
R+4h	Current Command	7.1.2.15
R+5h - R+7Bh	Reserved	
R+7Ch	Object Write	7.1.2.16
R+7Dh	Object Group	7.1.2.17
R+7Eh	Session	7.1.2.18
R+7Fh	Logical Unit	7.1.2.19

7.1.2.2 Attribute number 0h in all attributes pages

With the exception of the Root Directory and Group Directory attributes pages, all attributes pages defined by this standard or any other document shall contain an identification of the page in attribute number 0h. The format of the page identification shall be a 40 byte fixed length value with the format shown in table 56.

Table 56 — Attribute number 0h format for all attributes pages

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			VENDOR IDENTIFICATION				
7								(LSB)
8	(MSB)			ATTRIBUTES PAGE IDENTIFICATION				
39								(LSB)

The VENDOR IDENTIFICATION field shall contain eight bytes of ASCII data (i.e., code values 20h through 7Eh) identifying the organization that has defined the contents of the attribute page. Any unused bytes shall be placed at the end of the field (highest offset) and shall be filled with space characters (20h). The format of the VENDOR IDENTIFICATION field is identical to the format of the vendor identification field in the standard INQUIRY data (see SPC-3).

NOTE 5 It is intended that the VENDOR IDENTIFICATION field provide a unique identification of the organization that defined the attribute page contents. In the absence of a formal registration procedure, T10 maintains a list of vendor identification codes in use (see SPC-3). Organizations are requested to voluntarily submit their identification codes to T10 to prevent duplication of codes. The T10 web site, www.t10.org, provides a convenient means to request an identification code.

The ATTRIBUTES PAGE IDENTIFICATION field shall contain 32 bytes of ASCII data (i.e., code values 20h through 7Eh) identifying the attributes page in which it appears. Any unused bytes shall be placed at the end of the field (highest offset) and shall be filled with null characters (00h).

If the VENDOR IDENTIFICATION field contains INCITS, the first characters in the ATTRIBUTES PAGE IDENTIFICATION field shall identify the INCITS technical committee that has defined the contents of the attributes page (e.g., attributes pages defined by this standard have T10 as the first characters in the ATTRIBUTES PAGE IDENTIFICATION field).

7.1.2.3 Root Directory attributes page

The Root Directory attributes page (number R+0h) shall contain one attribute for every root, group, and user attribute page number accessible via the logical unit. Within the Root Directory attributes page, the attribute number of each attribute shall be equal to the page number of the accessible attributes page that it represents. The value of each attribute in the Root Directory attributes page shall be equal to the value of the attribute numbered 0h in the accessible attributes page that it represents.

As an example, table 57 shows the attributes in the Root Directory attributes page when only the attributes pages defined in this standard are accessible via the logical unit.

Table 57 — Example Root Directory attributes page contents

Attribute Number	Attribute Value
0h	INCITS T10 User Object Directory
1h	INCITS T10 User Object Information
2h	INCITS T10 User Object Resources
3h	INCITS T10 User Object Timestamps
G+0h	INCITS T10 Group Directory
G+1h	INCITS T10 Group Information
G+2h	INCITS T10 Group Resources
G+3h	INCITS T10 Group Timestamps
R+0h	INCITS T10 Root Directory
R+1h	INCITS T10 Root Information
R+2h	INCITS T10 Root Resources
R+3h	INCITS T10 Root Timestamps
R+4h	INCITS T10 Current Command
R+7Ch	INCITS T10 Object Write
R+7Dh	INCITS T10 Object Group
R+7Eh	INCITS T10 Session
R+7Fh	INCITS T10 Logical Unit

The contents of the Root Directory attributes page shall be maintained by the OSD.

If a set attributes parameter list contains an entry specifying the Root Directory attributes by page number R+0h, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL

REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the Root Directory attributes by page number R+0h, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

7.1.2.4 Group Directory attributes page

The Group Directory attributes page (number G+0h) shall contain one attribute for every group and user attribute page number accessible to the group or any user object within the group. Within the Group Directory attributes page, the attribute number of each attribute shall be equal to the page number of the accessible attributes page that it represents. The value of each attribute in the Group Directory attributes page shall be equal to the value of the attribute numbered 0h in the accessible attributes page that it represents.

The contents of the Group Directory attributes page shall be maintained by the OSD.

If a set attributes parameter list contains an entry specifying the Group Directory attributes by page number G+0h, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the Group Directory attributes by page number G+0h, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

7.1.2.5 User Object Directory attributes page

The User Object Directory attributes page (number 0h) shall contain one attribute for every user attribute page number accessible to the user object. Within the User Object Directory attributes page, the attribute number of each attribute shall be equal to the page number of the accessible attributes page that it represents. The value of each attribute in the User Object Directory attributes page shall be equal to the value of the attribute numbered 0h in the accessible attributes page that it represents.

The contents of the User Object Directory attributes page shall be maintained by the OSD.

If a set attributes parameter list contains an entry specifying the User Object Directory attributes by page number 0h, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB get or set attributes field specifies the User Object Directory attributes by page number 0h, the User Object Directory attributes shall not be retrieved or set and any associated set attribute number shall be ignored, this shall not be considered an error.

7.1.2.6 Root Information attributes page

The Root Information attributes page (number R+1h) shall contain the attributes listed in table 58.

Table 58 — Root Information attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h - 2h		Reserved	No	Yes
3h	8	Vendor identification	No	Yes
4h	16	Product identification	No	Yes
5h	32	Product model	No	Yes
6h	4	Product revision level	No	Yes
7h	8 or 16	Logical unit device identifier	No	Yes
8h	8	Used capacity	No	Yes
9h	8	Clock	Yes	Yes
Ah	variable	Serial number	No	Yes
Bh	variable	OSD name	Yes	No
Ch	variable	Username	Yes	No
Dh - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Root Information.

The vendor identification attribute (number 3h) shall contain the vendor identification of the manufacturer of the OSD in the same format as the VENDOR IDENTIFICATION field in the standard INQUIRY data (see SPC-3).

The product identification attribute (number 4h) shall contain the product identification of the OSD device in the same format as the PRODUCT IDENTIFICATION field in the standard INQUIRY data (see SPC-3).

The product model attribute (number 5h) shall contain 32 bytes of ASCII data (i.e., code values 20h through 7Eh) identifying the model of the OSD device. Any unused bytes shall be placed at the end of the attribute value (highest offset) and shall be filled with spaces characters (20h).

The product revision level attribute (number 6h) shall contain the product revision level of the OSD device in the same format as the PRODUCT REVISION LEVEL field in the standard INQUIRY data (see SPC-3).

The logical unit identifier attribute (number 7h) shall contain the IDENTIFIER field contents from an entry in the Device Identification VPD page (see SPC-3) whose the IDENTIFIER TYPE field contains 2h or 3h and whose ASSOCIATION field contains 0h.

The used capacity attribute (number 8h) shall contain the number of bytes used by the root object including attributes bytes represented as an unsigned binary value.

The clock attribute (number 9h) shall contain the current time in use by the OSD represented as an unsigned binary value that is the count of the number of seconds elapsed since January 1, 1970. The value shall be set to the correct Greenwich Mean Time when the OSD is manufactured and may be modified by the application client after that.

The serial number attribute (number Ah) shall contain the product serial number of the OSD in the same format as the PRODUCT SERIAL NUMBER field in the Unit Serial Number VPD page (see SPC-3).

The OSD name attribute (number Bh) shall contain an identification of the OSD device specified by the application client. The number of bytes in the OSD name attribute shall be set to zero by an OSD FORMAT service action (see 6.8).

The username attribute (number Ch) shall contain an identification of the user of the OSD device specified by the application client. The number of bytes in the username attribute shall be set to zero by an OSD FORMAT service action (see 6.8).

If a set attributes parameter list contains an entry specifying the number of an attribute that table 58 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 58 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

No page format is specified for the Root Information attributes page. If a CDB get or set attributes field specifies the page number of the Root Information attributes page, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

7.1.2.7 Group Information attributes page

The Group Information attributes page (number G+1h) shall contain the attributes listed in table 59.

Table 59 — Group Information attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	4	Object_Group_ID	No	Yes
2h - 7h		Reserved	No	
8h	8	Used capacity	No	Yes
9h - Bh		Reserved	No	
Ch	variable	Username	Yes	No
Dh - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Group Information.

The Object_Group_ID attribute (number 1h) shall contain the Object_Group_ID of the object group with which the Group Information attributes page is associated.

The used capacity attribute (number 8h) shall contain the number of bytes used by the group object and all user objects within the group including attributes bytes represented as an unsigned binary value.

The username attribute (number 10h) shall contain an identification of the user of the group specified by the application client. A CREATE OBJECT GROUP service action (see 6.6) shall copy the username attribute from the Root Information attributes page (see 7.1.2.6) to the new Group Information attributes page.

If a set attributes parameter list contains an entry specifying the number of an attribute that table 59 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 59 states may not be set, the command shall be terminated with a

CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

No page format is specified for the Group Information attributes page. If a CDB get or set attributes field specifies the page number of the Group Information attributes page, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

7.1.2.8 User Object Information attributes page

The User Object Information attributes page (number 1h) shall contain the attributes listed in table 60.

Table 60 — User Object Information attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	4	Object_Group_ID	No	Yes
2h	8	User_Object_ID	No	Yes
3h - 7h		Reserved	No	
8h	8	Used capacity	No	Yes
9h - Bh		Reserved	No	
Ch	variable	Username	Yes	No
Dh - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 User Object Information.

The Object_Group_ID attribute (number 1h) shall contain the Object_Group_ID of the user object with which the User Object Information attributes page is associated.

The User_Object_ID attribute (number 2h) shall contain the User_Object_ID of the user object with which the User Object Information attributes page is associated.

The used capacity attribute (number 8h) shall contain the number of bytes used by the user object including attributes bytes represented as an unsigned binary value.

The username attribute (number 10h) shall contain an identification of the user for the user object specified by the application client. A CREATE service action (see 6.4) or CREATE AND WRITE service action (see 6.5) shall copy the username attribute from the Group Information attributes page (see 7.1.2.7) to the new User Object Information attributes page.

If a set attributes parameter list contains an entry specifying the number of an attribute that table 60 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 60 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

No page format is specified for the User Object Information attributes page. If a CDB get or set attributes field specifies the page number of the User Object Information attributes page, the command shall be terminated with a

CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

7.1.2.9 Root Resources attributes page

The Root Resources attributes page (number R+2h) shall contain the attributes listed in table 61.

Table 61 — Root Resources attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	8	Object group capacity quota	Yes	No
2h	8	User object capacity quota	Yes	No
3h	4	User objects per group	Yes	No
4h	4	Group count	Yes	No
5h	8	Total capacity	No	Yes
6h	8	Remaining capacity	No	Yes
7h - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Root Resources.

The object group capacity quota attribute (number 1h) specifies the value to be placed in the capacity quota attribute of each newly created object group. The object group capacity quota attribute shall be set to FFFF FFFF FFFF FFFFh by an OSD FORMAT service action (see 6.8).

The user object capacity quota attribute (number 2h) specifies the value to be placed in the user object capacity quota attribute of each newly created object group. The user object capacity quota attribute shall be set to FFFF FFFF FFFF FFFFh by an OSD FORMAT service action.

The user objects per group attribute (number 3h) specifies the maximum number of user objects that may be created in an object group. The user objects per group attribute shall be set to FFFF FFFFh by an OSD FORMAT service action.

The group count attribute (number 4h) specifies the maximum number of object groups that may be created on the OSD logical unit. The group count attribute shall be set to FFFF FFFFh by an OSD FORMAT service action.

The total capacity attribute (number 5h) shall contain the total number of bytes on the OSD logical unit.

The remaining capacity attribute (number 6h) shall contain the number of bytes currently available for allocation on the OSD logical unit.

If a set attributes parameter list contains an entry specifying the number of an attribute that table 61 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 61 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The page format for the Root Resources attributes page is shown in table 62.

Table 62 — Root Resources attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			PAGE NUMBER (R+2h)				
3								(LSB)
4	(MSB)			PAGE LENGTH (28h)				
7								(LSB)
8	(MSB)			OBJECT GROUP CAPACITY QUOTA				
15								(LSB)
16	(MSB)			USER OBJECT CAPACITY QUOTA				
23								(LSB)
24	(MSB)			USER OBJECTS PER GROUP				
27								(LSB)
28	(MSB)			GROUP COUNT				
31								(LSB)
32	(MSB)			TOTAL CAPACITY				
39								(LSB)
40	(MSB)			REMAINING CAPACITY				
47								(LSB)

The page number field contains the attribute page number of the Root Resources attributes page.

The page length field contains the number of additional bytes in the page format of the Root Resources attributes page.

The OBJECT GROUP CAPACITY QUOTA field contains the value of the object group capacity quota attribute.

The USER OBJECT CAPACITY QUOTA field contains the value of the user object capacity quota attribute.

The USER OBJECTS PER GROUP field contains the value of the user objects per group attribute.

The GROUP COUNT field contains the value of the group count attribute.

The TOTAL CAPACITY field contains the value of the total capacity attribute.

The REMAINING CAPACITY field contains the value of the remaining capacity attribute.

7.1.2.10 Group Resources attributes page

The Group Resources attributes page (number G+2h) shall contain the attributes listed in table 63.

Table 63 — Group Resources attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	8	Capacity quota	Yes	No
2h	8	User object capacity quota	Yes	No
3h	4	Object count	Yes	No
4h - 5h		Reserved	No	
6h	8	Remaining capacity	No	Yes
7h - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Group Resources.

The capacity quota attribute (number 1h) specifies the maximum value the OSD shall allow in the used capacity attribute of the Group Information attributes page (see 7.1.2.7). A CREATE OBJECT GROUP service action (see 6.6) shall copy the object group capacity quota attribute from the Root Resources attributes page (see 7.1.2.9) to the capacity quota attribute in new Group Resources attributes page.

The user object capacity quota attribute (number 2h) specifies the value to be placed in the capacity quota attribute of each newly created user object. A CREATE OBJECT GROUP service action (see 6.6) shall copy the user object capacity quota attribute from the Root Resources attributes page to the new Group Resources attributes page.

The object count attribute (number 3h) specifies the maximum number of user objects that may be created in the object group. A CREATE OBJECT GROUP service action (see 6.6) shall copy the user objects per group attribute from the Root Resources attributes page to the object count attribute in new Group Resources attributes page.

The remaining capacity attribute (number 6h) indicates the number of unused bytes available to the object group and user objects within the object group. The value shall be the smaller of:

a) The value in the capacity quota attribute minus the value in the used capacity attribute in the Group Information attributes page; or
b) The value in the total capacity attribute in the Root Resources attributes page.

If a set attributes parameter list contains an entry specifying the number of an attribute that table 63 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 63 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The page format for the Group Resources attributes page is shown in table 64.

Table 64 — Group Resources attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				PAGE NUMBER (G+2h)				(LSB)
4	(MSB)							
7				PAGE LENGTH (1Ch)				(LSB)
8	(MSB)							
15				CAPACITY QUOTA				(LSB)
16	(MSB)							
23				USER OBJECT CAPACITY QUOTA				(LSB)
24	(MSB)							
27				OBJECT COUNT				(LSB)
28	(MSB)							
35				REMAINING CAPACITY				(LSB)

The page number field contains the attribute page number of the Group Resources attributes page.

The page length field contains the number of additional bytes in the page format of the Group Resources attributes page.

The CAPACITY QUOTA field contains the value of the capacity quota attribute.

The USER OBJECT CAPACITY QUOTA field contains the value of the user object capacity quota attribute.

The OBJECT COUNT field contains the value of the object count attribute.

The REMAINING CAPACITY field contains the value of the remaining capacity attribute.

7.1.2.11 User Object Resources attributes page

The User Object Resources attributes page (number 2h) shall contain the attributes listed in table 65.

Table 65 — User Object Resources attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	8	Capacity quota	Yes	No
2h - 5h		Reserved	No	
6h	8	Remaining capacity	No	Yes
7h	8	Object logical length	No	Yes
8h	8	Starting byte address of write or append	No	Yes
9h - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 User Object Resources.

The capacity quota attribute (number 1h) specifies the maximum value the OSD shall allow in the used capacity attribute of the User Object Information attributes page (see 7.1.2.8). A CREATE service action (see 6.4) or CREATE AND WRITE service action (see 6.5) shall copy the user object capacity quota attribute from the Group Resources attributes page (see 7.1.2.10) to the capacity quota attribute in new User Object Resources attributes page.

The remaining capacity attribute (number 6h) specifies the number of unused bytes available to the user object. The value shall be the smaller of:

a) The value in the capacity quota attribute minus the value in the used capacity attribute of the User Object Information attributes page;
b) The value in the remaining capacity attribute in the Group Resources attributes page; or
c) The value in the total capacity attribute in the Root Resources attributes page (see 7.1.2.9).

The object logical length attribute (number 7h) specifies the largest valued byte number written in the associated user object.

The starting byte address of write or append attribute (number 8h) specifies the starting byte address specified by the most recent CREATE AND WRITE service action (see 6.5) or WRITE service action (see 6.19) or provided by the OSD for an APPEND service action (see 6.2).

If a set attributes parameter list contains an entry specifying the number of an attribute that table 65 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 65 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The page format for the User Object Resources attributes page is shown in table 64.

Table 66 — User Object Resources attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				PAGE NUMBER (2h)				(LSB)
4	(MSB)							
7				PAGE LENGTH (16h)				(LSB)
8	(MSB)							
15				CAPACITY QUOTA				(LSB)
16	(MSB)							
23				REMAINING CAPACITY				(LSB)
24	(MSB)							
31				OBJECT LOGICAL LENGTH				(LSB)
32	(MSB)							
39				STARTING BYTE ADDRESS OF WRITE OR APPEND				(LSB)

The page length field contains the number of additional bytes in the page format of the User Object Resources attributes page.

The CAPACITY QUOTA field contains the value of the capacity quota attribute.

The REMAINING CAPACITY field contains the value of the remaining capacity attribute.

The OBJECT LOGICAL LENGTH field contains the value of the object logical length attribute.

The STARTING BYTE ADDRESS OF WRITE OR APPEND field contains the value of the starting byte address of write or append attribute.

7.1.2.12 Root Timestamps attributes page

The Root Timestamps attributes page (number R+3h) shall contain the attributes listed in table 67.

Table 67 — Root Timestamps attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	8	Created time	No	Yes
2h	8	Attributes accessed time	No	Yes
3h	8	Attributes modified time	No	Yes
4h	8	Data accessed time	No	Yes
5h	8	Data modified time	No	Yes
6h - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Root Timestamps.

The created time attribute (number 1h) shall contain the value of the clock attribute in the Root Information attributes page (see 7.1.2.6) at the completion of the most recent OSD FORMAT service action (see 6.8).

The attributes accessed time attribute (number 2h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent service action that retrieved any attributes page associated with the root object.

The attributes modified time attribute (number 3h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent service action that set any attributes page associated with the root object.

The data accessed time attribute (number 4h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent LIST service action (see 6.12) that listed the object groups in the root object.

The data modified time attribute (number 5h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent CREATE OBJECT GROUP service action (see 6.6).

If a set attributes parameter list contains an entry specifying the number of an attribute that table 67 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 67 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The page format for the Root Timestamps attributes page is shown in table 68.

Table 68 — Root Timestamps attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				PAGE NUMBER (R+3h)				(LSB)
4	(MSB)							
7				PAGE LENGTH (28h)				(LSB)
8	(MSB)							
15				CREATED TIME				(LSB)
16	(MSB)							
23				ATTRIBUTES ACCESSED TIME				(LSB)
24	(MSB)							
31				ATTRIBUTES MODIFIED TIME				(LSB)
32	(MSB)							
39				DATA ACCESSED TIME				(LSB)
40	(MSB)							
47				DATA MODIFIED TIME				(LSB)

The page number field contains the attribute page number of the Root Timestamps attributes page.

The page length field contains the number of additional bytes in the page format of the Root Timestamps attributes page.

The CREATED TIME field contains the value of the created time attribute.

The ATTRIBUTES ACCESSED TIME field contains the value of the attributes accessed time attribute.

The ATTRIBUTES MODIFIED TIME field contains the value of the attributes modified time attribute.

The DATA ACCESSED TIME field contains the value of the data accessed time attribute.

The DATA MODIFIED TIME field contains the value of the data modified time attribute.

7.1.2.13 Group Timestamps attributes page

The Group Timestamps attributes page (number G+3h) shall contain the attributes listed in table 69.

Table 69 — Group Timestamps attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	8	Created time	No	Yes
2h	8	Attributes accessed time	No	Yes
3h	8	Attributes modified time	No	Yes
4h	8	Data accessed time	No	Yes
5h	8	Data modified time	No	Yes
6h - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Group Timestamps.

The created time attribute (number 1h) shall contain the value of the clock attribute in the Root Information attributes page (see 7.1.2.6) at the completion of the CREATE OBJECT GROUP service action (see 6.6) that created the object group.

The attributes accessed time attribute (number 2h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent service action that retrieved any attributes page associated with the group object.

The attributes modified time attribute (number 3h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent service action that set any attributes page associated with the object group.

The data accessed time attribute (number 4h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent LIST service action (see 6.12) that listed the user objects in the group object.

The data modified time attribute (number 5h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent CREATE service action (see 6.4) or CREATE AND WRITE service action (see 6.5) that created a user object in the object group.

If a set attributes parameter list contains an entry specifying the number of an attribute that table 69 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 69 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The page format for the Group Timestamps attributes page is shown in table 70.

Table 70 — Group Timestamps attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			PAGE NUMBER (G+3h)				
3								(LSB)
4	(MSB)			PAGE LENGTH (28h)				
7								(LSB)
8	(MSB)			CREATED TIME				
15								(LSB)
16	(MSB)			ATTRIBUTES ACCESSED TIME				
23								(LSB)
24	(MSB)			ATTRIBUTES MODIFIED TIME				
31								(LSB)
32	(MSB)			DATA ACCESSED TIME				
39								(LSB)
40	(MSB)			DATA MODIFIED TIME				
47								(LSB)

The page number field contains the attribute page number of the Group Timestamps attributes page.

The page length field contains the number of additional bytes in the page format of the Group Timestamps attributes page.

The CREATED TIME field contains the value of the created time attribute.

The ATTRIBUTES ACCESSED TIME field contains the value of the attributes accessed time attribute.

The ATTRIBUTES MODIFIED TIME field contains the value of the attributes modified time attribute.

The DATA ACCESSED TIME field contains the value of the data accessed time attribute.

The DATA MODIFIED TIME field contains the value of the data modified time attribute.

7.1.2.14 User Object Timestamps attributes page

The User Object Timestamps attributes page (number 3h) shall contain the attributes listed in table 71.

Table 71 — User Object Timestamps attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	8	Created time	No	Yes
2h	8	Attributes accessed time	No	Yes
3h	8	Attributes modified time	No	Yes
4h	8	Data accessed time	No	Yes
5h	8	Data modified time	No	Yes
6h - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 User Object Timestamps.

The created time attribute (number 1h) shall contain the value of the clock attribute in the Root Root Information attributes page (see 7.1.2.6) at the completion of the CREATE service action (see 6.4) or CREATE AND WRITE service action (see 6.5) that created the associated user object.

The attributes accessed time attribute (number 2h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent service action that retrieved any attributes page associated with the user object.

The attributes modified time attribute (number 3h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent service action that set any attributes page associated with the user object.

The data accessed time attribute (number 4h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent READ service action (see 6.14) that accessed the user object.

The data modified time attribute (number 5h) shall contain the value of the clock attribute in the Root Information attributes page at the completion of the most recent WRITE service action (see 6.19), APPEND service action (see 6.2), or CREATE AND WRITE service action (see 6.5) that stored data in the user object.

If a set attributes parameter list contains an entry specifying the number of an attribute that table 71 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 71 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The page format for the User Object Timestamps attributes page is shown in table 72.

Table 72 — User Object Timestamps attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				PAGE NUMBER (3h)				(LSB)
4	(MSB)							
7				PAGE LENGTH (28h)				(LSB)
8	(MSB)							
15				CREATED TIME				(LSB)
16	(MSB)							
23				ATTRIBUTES ACCESSED TIME				(LSB)
24	(MSB)							
31				ATTRIBUTES MODIFIED TIME				(LSB)
32	(MSB)							
39				DATA ACCESSED TIME				(LSB)
40	(MSB)							
47				DATA MODIFIED TIME				(LSB)

The page number field contains the attribute page number of the User Object Timestamps attributes page.

The page length field contains the number of additional bytes in the page format of the User Object Timestamps attributes page.

The CREATED TIME field contains the value of the created time attribute.

The ATTRIBUTES ACCESSED TIME field contains the value of the attributes accessed time attribute.

The ATTRIBUTES MODIFIED TIME field contains the value of the attributes modified time attribute.

The DATA ACCESSED TIME field contains the value of the data accessed time attribute.

The DATA MODIFIED TIME field contains the value of the data modified time attribute.

7.1.2.15 Current Command attributes page

The Current Command attributes page (number R+4h) shall contain the attributes listed in table 73.

Table 73 — Current Command attributes page contents

Attribute Number	Bytes	Attribute	May be set	OSD Provided
0h	40	Page identification	No	Yes
1h	4	Object_Session_ID	No	Yes
2h	4	Assigned Attribute Page Number	No	Yes
3h - FFFF FFFFh		Reserved	No	

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Current Command.

The Object_Session_ID attribute (number 1h) shall contain the Object_Session_ID of the object session that is being used by the current command. If there is no object session is being used by the current command, the Object_Session_ID attribute value shall be zero.

When the current command is a CREATE ATTRIBUTES PAGE service action (see 6.6), the assigned attribute page number attribute (number 2h) shall contain the assigned attribute page number. Otherwise, the assigned attribute page number attribute shall contain zero.

If a set attributes parameter list contains an entry specifying the number of an attribute that table 73 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. If a CDB set attributes field specifies the number of an attribute that table 73 states may not be set, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The page format for the Current Command attributes page is shown in table 74.

Table 74 — Current Command attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			PAGE NUMBER (R+4h)				
3								(LSB)
4	(MSB)			PAGE LENGTH (8h)				
7								(LSB)
8	(MSB)			OBJECT_SESSION_ID				
11								(LSB)
12	(MSB)			ASSIGNED ATTRIBUTE PAGE NUMBER				
15								(LSB)

The page number field contains the attribute page number of the Current Command attributes page.

The page length field contains the number of additional bytes in the page format of the Current Command attributes page.

The OBJECT_SESSION_ID field contains the value of the Object_Session_ID attribute.

The ASSIGNED ATTRIBUTE PAGE NUMBER field contains the value of the assigned attribute page number attribute.

7.1.2.16 Object Write attributes page

Editors Note 20 - ROW: The Object Write attributes page provides response data equal to or in excess of the response data specified for the APPEND, CREATE, CREATE AND WRITE, IMPORT USER OBJECT, and WRITE service actions in OSD r05.

The Object Write attributes page (number R+7Ch) shall reference the attributes listed in table 75.

Table 75 — Object Write attributes page attributes references

Attribute Number	Bytes	Name in This Attributes Page for Referenced Attribute	Referenced Attribute		
			Page	Number	Reference
0h	40	Page identification	Not a reference		
1h	4	Object_Group_ID	1h	1h	7.1.2.8
2h	8	User_Object_ID	1h	2h	7.1.2.8
3h	4	Object_Session_ID	R+4h	1h	7.1.2.15
4h	8	User object logical length	2h	7h	7.1.2.11
5h	8	User object remaining capacity	2h	6h	7.1.2.11
6h	8	Starting byte address of write or append	2h	8h	7.1.2.11
7h - FFFF FFFFh	Reserved				

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Object Write.

The page format for the Object Write attributes page is shown in table 74.

Table 76 — Object Write attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				PAGE NUMBER (R+7Ch)				(LSB)
4	(MSB)							
7				PAGE LENGTH (28h)				(LSB)
8	(MSB)							
11				OBJECT_GROUP_ID				(LSB)
12	(MSB)							
19				USER_OBJECT_ID				(LSB)
20	(MSB)							
23				OBJECT_SESSION_ID				(LSB)
24	(MSB)							
31				USER OBJECT LOGICAL LENGTH				(LSB)
32	(MSB)							
39				USER OBJECT REMAINING CAPACITY				(LSB)
40	(MSB)							
47				STARTING BYTE ADDRESS OF WRITE OR APPEND				(LSB)

The page number field contains the attribute page number of the Object Write attributes page.

The page length field contains the number of additional bytes in the page format of the Object Write attributes page.

The OBJECT_GROUP_ID field contains the value of the Object_Group_ID attribute from the User Object Information attributes page (see 7.1.2.8).

The USER_OBJECT_ID field contains the value of the User_Object_ID attribute from the User Object Information attributes page.

The OBJECT_SESSION_ID field contains the value of the Object_Session_ID attribute from the Current Command attributes page (see 7.1.2.15).

The USER OBJECT LOGICAL LENGTH field contains the value of the object logical length attribute from the User Object Resources attributes page (see 7.1.2.11).

The USER OBJECT REMAINING CAPACITY field contains the value of the remaining capacity attribute from the User Object Resources attributes page.

The STARTING BYTE ADDRESS OF WRITE OR APPEND field contains the value of the starting byte address of write or append attribute from the User Object Resources attributes page.

7.1.2.17 Object Group attributes page

Editors Note 21 - ROW: The Object Group attributes page provides response data equal to or in excess of the response data specified for the CREATE OBJECT GROUP and REMOVE (user object) service actions in OSD r05.

The Object Group attributes page (number R+7Dh) shall reference the attributes listed in table 77.

Table 77 — Object Group attributes page attributes references

Attribute Number	Bytes	Name in This Attributes Page for Referenced Attribute	Referenced Attribute		
			Page	Number	Reference
0h	40	Page identification	Not a reference		
1h	4	Object_Group_ID	G+1h	1h	7.1.2.7
2h	8	Object group remaining capacity	G+2h	6h	7.1.2.10
3h - FFFF FFFFh	Reserved				

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Object Group.

The page format for the Object Group attributes page is shown in table 76.

Table 78 — Object Group attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			PAGE NUMBER (R+7Dh)				
3								(LSB)
4	(MSB)			PAGE LENGTH (0Ch)				
7								(LSB)
8	(MSB)			OBJECT_GROUP_ID				
11								(LSB)
12	(MSB)			OBJECT GROUP REMAINING CAPACITY				
19								(LSB)

The page number field contains the attribute page number of the Object Group attributes page.

The page length field contains the number of additional bytes in the page format of the Object Group attributes page.

The OBJECT_GROUP_ID field contains the value of the Object_Group_ID attribute from the Group Information attributes page (see 7.1.2.7).

The OBJECT GROUP REMAINING CAPACITY field contains the value of the remaining capacity attribute from the Group Resources attributes page (see 7.1.2.10).

7.1.2.18 Session attributes page

Editors Note 22 - ROW: The Session attributes page provides response data equal to or in excess of the response data specified for the OPEN service action in OSD r05.

The Session attributes page (number R+7Eh) shall reference the attributes listed in table 79.

Table 79 — Session attributes page attributes references

Attribute Number	Bytes	Name in This Attributes Page for Referenced Attribute	Referenced Attribute		
			Page	Number	Reference
0h	40	Page identification	Not a reference		
1h	4	Object_Group_ID	1h	1h	7.1.2.8
2h	8	User_Object_ID	1h	2h	7.1.2.8
3h	4	Object_Session_ID	R+4h	1h	7.1.2.15
4h	8	User object logical length	2h	7h	7.1.2.11
5h	8	User object remaining capacity	2h	6h	7.1.2.11
6h - FFFF FFFFh	Reserved				

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Session.

The page format for the Session attributes page is shown in table 80.

Table 80 — Session attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			PAGE NUMBER (R+7Eh)				
3								(LSB)
4	(MSB)			PAGE LENGTH (20h)				
7								(LSB)
8	(MSB)			OBJECT_GROUP_ID				
11								(LSB)
12	(MSB)			USER_OBJECT_ID				
19								(LSB)
20	(MSB)			OBJECT_SESSION_ID				
23								(LSB)
24	(MSB)			USER OBJECT LOGICAL LENGTH				
31								(LSB)
32	(MSB)			USER OBJECT REMAINING CAPACITY				
39								(LSB)

The page number field contains the attribute page number of the Session attributes page.

The page length field contains the number of additional bytes in the page format of the Session attributes page.

The OBJECT_GROUP_ID field contains the value of the Object_Group_ID attribute from the User Object Information attributes page (see 7.1.2.8).

The USER_OBJECT_ID field contains the value of the User_Object_ID attribute from the User Object Information attributes page.

The OBJECT_SESSION_ID field contains the value of the Object_Session_ID attribute from the Current Command attributes page (see 7.1.2.15).

The USER OBJECT LOGICAL LENGTH field contains the value of the object logical length attribute from the User Object Resources attributes page (see 7.1.2.11).

The USER OBJECT REMAINING CAPACITY field contains the value of the remaining capacity attribute from the User Object Resources attributes page.

7.1.2.19 Logical Unit attributes page

Editors Note 23 - ROW: The Logical Unit attributes page provides response data equal to or in excess of the response data specified for the FORMAT OSD and REMOVE GROUP OBJECT service actions in OSD r05.

The Logical Unit attributes page (number R+7Fh) shall reference the attributes listed in table 81.

Table 81 — Logical Unit attributes page attributes references

Attribute Number	Bytes	Name in This Attributes Page for Referenced Attribute	Referenced Attribute		
			Page	Number	Reference
0h	40	Page identification	Not a reference		
1h	8	Logical unit remaining capacity	R+2h	6h	7.1.2.9
2h - FFFF FFFFh	Reserved				

The page identification attribute (number 0h) shall have the format described in 7.1.2.2 with the VENDOR IDENTIFICATION field containing INCITS and the ATTRIBUTE PAGE IDENTIFICATION field containing T10 Logical Unit.

The page format for the Logical Unit attributes page is shown in table 82.

Table 82 — Logical Unit attributes page format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			PAGE NUMBER (R+7Fh)				
3								(LSB)
4	(MSB)			PAGE LENGTH (08h)				
7								(LSB)
8	(MSB)			LOGICAL UNIT REMAINING CAPACITY				
15								(LSB)

The page number field contains the attribute page number of the Logical Unit attributes page.

The page length field contains the number of additional bytes in the page format of the Logical Unit attributes page.

The LOGICAL UNIT REMAINING CAPACITY field contains the value of the remaining capacity attribute from the Root Resources attributes page (see 7.1.2.9).

7.1.3 OSD attribute lists

7.1.3.1 Attribute lists overview

An attribute list acts on one or more individual attribute values using attribute page, attribute number, and optionally Object_Group_ID and User_Object_ID values to specify the attribute values to be retrieved or set.

The format of an attribute list is shown in table 83.

Table 83 — Attribute list format

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved				LIST TYPE			
1	Reserved							
2	(MSB)			LIST LENGTH (n-3)				
3								(LSB)
	Attribute list entries							
4				Attribute list entry 0				
	. . .							
n	Attribute list entry x							

The LIST TYPE field (see table 84) specifies the format of all attribute list entries in the attribute list.

Table 84 — List type values

List Type	Description	Support Requirement	Reference
0h	Reserved		
1h	Retrieve attributes for this object	Mandatory	7.1.3.2
2h	Retrieve attributes for any object	Optional	7.1.3.3
3h	Retrieve attribute references	Optional	7.1.3.4
4h - 8h	Reserved		
9h	Retrieved/Set attributes for this object	Mandatory	7.1.3.5
Ah	Retrieved/Set attributes for any object	Optional	7.1.3.6
Bh - Fh			

The types of lists retrieved are shown in table 85.

Table 85 — Retrieved list types

Get List Type	Reference	Description	Returned List Type	Reference
1h	7.1.3.2	Get attributes for this object only	9h	7.1.3.5
2h	7.1.3.3	Get attributes for any object	Ah	7.1.3.6
3h	7.1.3.4	Get reference attributes when only this object is referenced	1h	7.1.3.2
		Get reference attributes when any object is referenced	2h	7.1.3.3

The LIST LENGTH field indicates the number of bytes of attribute list entries that follow. When attributes list parameter data is sent from the application client to the device server, the LIST LENGTH field may contain zero; the device server shall use the length of the list specified in the CDB and shall ignore the contents of the LIST LENGTH field.

7.1.3.2 List entry format for retrieving attributes for this object

The attribute list entry format shown in table 86 is used for specifying the attributes to be retrieved by a GET ATTRIBUTES service action (see 6.10) or equivalent operation and for specifying the references to other attributes for a CREATE ATTRIBUTES PAGE service action (see 6.6) that defines or replaces an attributes page containing only references to attributes in other attribute pages.

Table 86 — List entry format for retrieving attributes for this object

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3	ATTRIBUTE PAGE							(LSB)
4	(MSB)							
7	ATTRIBUTE NUMBER							(LSB)

The ATTRIBUTE PAGE field specifies that page number of one attribute to be returned. If the specified attribute page number is not associated with the user object specified by a service action, the object group of which that user

object is a member, or the root object the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The ATTRIBUTE NUMBER field specifies the attribute number within the attribute page specified by the ATTRIBUTE PAGE field of the one attribute value to be returned. If the attribute specified by the ATTRIBUTE PAGE and ATTRIBUTE NUMBER fields has no defined value, an attribute value having a length of zero shall be returned.

NOTE 6 Specifying attribute page and attribute numbers values of FFFF FFFFh causes all attributes values in all pages associated with the user object specified by a service action, the object group of which that user object is a member, and the root object to be returned. Specifying an attribute numbers value of FFFF FFFFh causes all attributes values in the specified attributes page to be returned.

When FFFF FFFFh is used as an attributes page number or attribute number value, only those attributes with non-zero lengths shall be returned.

7.1.3.3 List entry format for retrieving attributes for any object

The attribute list entry format shown in table 87 is used for specifying the attributes to be retrieved by a GET ATTRIBUTES service action (see 6.10) or equivalent operation and for specifying the references to other attributes for a CREATE ATTRIBUTES PAGE service action (see 6.6) that defines or replaces an attributes page containing only references to attributes in other attribute pages.

Table 87 — List entry format for retrieving attributes for any object

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			OBJECT_GROUP_ID				
3								(LSB)
4	(MSB)			USER_OBJECT_ID				
11								(LSB)
12	(MSB)			ATTRIBUTE PAGE				
15								(LSB)
16	(MSB)			ATTRIBUTE NUMBER				
19								(LSB)

The OBJECT_GROUP_ID field specifies the Object_Group_ID of the object whose associated attribute value is to be returned. If the specified Object_Group_ID is not defined in the OSD logical unit, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The USER_OBJECT_ID field specifies the User_Object_ID of the object whose associated attribute value is to be returned. If the specified User_Object_ID is not defined in the object group specified by the OBJECT_GROUP_ID field, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The ATTRIBUTE PAGE field specifies that page number of one attribute associated with the specified object to be returned. If the specified attribute page number is not associated with specified object, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The ATTRIBUTE NUMBER field specifies the attribute number within the attribute page specified by the OBJECT_GROUP_ID, USER_OBJECT_ID, and ATTRIBUTE PAGE fields of the one attribute value to be returned. If the attribute specified by the OBJECT_GROUP_ID, USER_OBJECT_ID, ATTRIBUTE PAGE, and ATTRIBUTE NUMBER fields has no defined value, an attribute value having a length of zero shall be returned.

NOTE 7 Specifying attribute page and attribute numbers values of FFFF FFFFh causes all attributes value in all pages associated with the specified object to be returned. Specifying an attribute numbers value of FFFF FFFFh causes all attributes values in the specified attributes page to be returned.

When FFFF FFFFh is used as an attributes page number or attribute number value, only those attributes with non-zero lengths shall be returned.

7.1.3.4 List entry format for retrieving attribute references

The attribute list entry format shown in table 88 is used for specifying the attribute references to be retrieved by a GET ATTRIBUTES service action (see 6.10).

Table 88 — List entry format for retrieving attribute references

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3	ATTRIBUTE PAGE							(LSB)
4	(MSB)							
7	ATTRIBUTE NUMBER							(LSB)

The ATTRIBUTE PAGE field specifies that page number of one attribute reference to be returned. If the specified attribute page is not a page containing references to other attributes or if the specified attribute page number is not associated with the user object specified by a service action, the object group of which that user object is a member, or the root object, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The ATTRIBUTE NUMBER field specifies the attribute number within the attribute page specified by the ATTRIBUTE PAGE field of the one attribute reference to be returned. If the attribute specified by the ATTRIBUTE PAGE and ATTRIBUTE NUMBER fields does not define a reference, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

NOTE 8 Specifying an attribute number of FFFF FFFFh causes all attributes references in the specified attributes page to be returned.

7.1.3.5 List entry format for retrieved attributes and for setting attributes for this object

The attribute list entry format shown in table 89 is used for returning the each attribute value to be retrieved by a GET ATTRIBUTES service action (see 6.10) and for specifying each attribute value to be set by a SET ATTRIBUTES service action (see 6.18) or equivalent operations.

Table 89 — List entry format for retrieved attributes and for setting attributes for this object

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				ATTRIBUTE PAGE				(LSB)
4	(MSB)							
7				ATTRIBUTE NUMBER				(LSB)
8	(MSB)							
9				ATTRIBUTE LENGTH (n-9)				(LSB)
10	(MSB)							
n				ATTRIBUTE VALUE				(LSB)

The ATTRIBUTE PAGE field specifies that page number of the attribute value. If the specified attribute page number is not associated with the user object specified by an attributes set operation, the object group of which that user object is a member, or the root object, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The ATTRIBUTE NUMBER field specifies the attribute number within the attribute page specified by the ATTRIBUTE PAGE field of the attribute value.

The ATTRIBUTE LENGTH field specifies the length of the attribute value in bytes.

The ATTRIBUTE VALUE field specifies the attribute value.

If the attribute specified by the ATTRIBUTE PAGE and ATTRIBUTE NUMBER fields in a set operation has a defined value, the value shall be replaced by the value specified by the ATTRIBUTE LENGTH and ATTRIBUTE VALUE fields. Otherwise, a new attribute shall be created with the attribute number specified by the ATTRIBUTE NUMBER field in the in the attribute page specified by the ATTRIBUTE PAGE field.

If the ATTRIBUTE PAGE or ATTRIBUTE NUMBER field contains FFFF FFFFh for a set operation, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

7.1.3.6 List entry format for retrieved attributes and for setting attributes for any object

The attribute list entry format shown in table 90 is used for returning the each attribute value to be retrieved by a GET ATTRIBUTES service action (see 6.10) and for specifying each attribute value to be set by a SET ATTRIBUTES service action (see 6.18) or equivalent operations.

Table 90 — List entry format for retrieved attributes and for setting attributes for any object

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				OBJECT_GROUP_ID				(LSB)
4	(MSB)							
11				USER_OBJECT_ID				(LSB)
12	(MSB)							
15				ATTRIBUTE PAGE				(LSB)
16	(MSB)							
19				ATTRIBUTE NUMBER				(LSB)
20	(MSB)							
21				ATTRIBUTE LENGTH (n-21)				(LSB)
22	(MSB)							
n				ATTRIBUTE VALUE				(LSB)

The OBJECT_GROUP_ID field specifies the Object_Group_ID of the object associated with the attribute value. If the specified Object_Group_ID for an attributes set operation is not defined in the OSD logical unit, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The USER_OBJECT_ID field specifies the User_Object_ID of the object within the object group specified by the OBJECT_GROUP_ID field associated with the attribute value. If the specified User_Object_ID for an attributes set operation is not defined in the object group specified by the OBJECT_GROUP_ID field, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The ATTRIBUTE PAGE field specifies that page number of the attribute value. If the specified attribute page number is not associated with the object specified by the OBJECT_GROUP_ID and USER_OBJECT_ID fields in an attributes set operation, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The ATTRIBUTE NUMBER field specifies the attribute number within the attribute page specified by the OBJECT_GROUP_ID, USER_OBJECT_ID, and ATTRIBUTE PAGE fields of the attribute value.

The ATTRIBUTE LENGTH field specifies the length of the attribute value in bytes.

The ATTRIBUTE VALUE field specifies the attribute value.

If the attribute specified by the OBJECT_GROUP_ID, USER_OBJECT_ID, ATTRIBUTE PAGE, and ATTRIBUTE NUMBER fields in a set operation has a defined value, the value shall be replaced by the value specified by the ATTRIBUTE LENGTH and ATTRIBUTE VALUE fields. Otherwise, a new attribute shall be created with the attribute number specified by the

ATTRIBUTE NUMBER field in the in the attribute page specified by the OBJECT_GROUP_ID, USER_OBJECT_ID, and ATTRIBUTE PAGE fields.

If the ATTRIBUTE PAGE or ATTRIBUTE NUMBER field contains FFFF FFFFh for a set operation, the command shall be terminated with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

7.2 Diagnostic parameters

This subclause defines the descriptors and pages for diagnostic parameters used with OSD type devices.

The diagnostic page codes for OSD devices are defined in table 91.

Table 91 — OSD diagnostic page codes

Page Code	Description	Reference
00h	Supported diagnostics pages	SPC-3
01h - 3Fh	Reserved (for pages that apply to all device types)	
40h - 7Fh	Reserved	
80h - FFh	Vendor specific pages	

7.3 Log parameters

This subclause defines the descriptors and pages for log parameters used with OSD type devices.

The log page codes for OSD devices are defined in table 92.

Table 92 — OSD log page codes

Page Code	Description	Reference
01h	Buffer over-run/under-run page	SPC-3
0Bh	Last *n* deferred errors or asynchronous events page	SPC-3
07h	Last *n* error events page	SPC-3
06h	Non-medium error page	SPC-3
00h	Supported log pages	SPC-3
02h - 05h	Reserved	
08h - 0Ah	Reserved	
0Ch - 2Fh	Reserved	
3Fh	Reserved	
30h - 3Eh	Vendor specific pages	

7.4 Vital product data parameters

This subclause defines the descriptors and pages for vital product data parameters used with OSD type devices.

The vital product data page codes for OSD devices are defined in table 93.

Table 93 — OSD vital product data page codes

Page code	Description	Reference	Support Requirements
82h	ASCII implemented operating definition page	SPC-3	Optional
01h - 7Fh	ASCII information page	SPC-3	Optional
83h	Device identification page	SPC-3	Mandatory
00h	Supported vital product data pages	SPC-3	Mandatory
80h	Unit serial number page	SPC-3	Optional
C0h - FFh	Vendor specific		

Annex A
(Informative)

Numeric order codes

A.1 Service action codes

The variable length CDB service action codes assigned by this standard are shown in table A.1.

Table A.1 — Numerical order OSD service action codes

Service Action	Command
8801h	FORMAT OSD
8802h	CREATE
8803h	LIST
8804h	OPEN
8805h	READ
8806h	WRITE
8807h	APPEND
8808h	FLUSh
8809h	CLOSE
880Ah	REMOVE
880Bh	CREATE OBJECT_GROUP
880Ch	REMOVE OBJECT_GROUP
880Dh	IMPORT USER_OBJECT
880Eh	GET ATTRIBUTES
880Fh	SET ATTRIBUTES
8810h	Reserved
8811h	OSD SYNCHRONIZE CACHE
8812h	CREATE AND WRITE
8813h	CREATE ATTRIBUTE PAGE
8814h	REMOVE ATTRIBUTE PAGE
8815h - 8FFFh	Reserved

Annex B
(Informative)

Research Notes

B.1 Overview

This annex attempts to capture some of the more important discussion concerning the commands. There was not unanimity in the above definitions, and the following may serve useful in helping a wider audience understand the rationale for the choices that were made.

Editors Note 24 - GEM: This annex should be deleted or set in a different vane prior to forwarding OSD.

Fields marked Done in this subclause have been incorporated into the main body text.

B.2 FORMAT OSD

It should not be possible to just start using a storage device in LBA mode after it has been initialized as an OSD device. To return a device to LBA mode after it has been initialized as an OSD device, a vendor specific service action (e.g., download microcode) needs to be taken. This may cause the entire contents of the storage device to be destroyed.

B.3 CREATE

Possible options:

Table B.1 — CREATE Option Byte 2 format

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	ATTR	Done	CMPL

CMPL is set to one when the CREATE service action is to cause a complete object to be created, written and closed. A data phase associated with the CREATE service action will convey the object data to the OSD device. The LENGTH field in the CREATE service action will contain the length of this data. This is intended to improve performance in environments where many small objects are being created.

ATTR is set to one to indicate that the information transfer includes attribute data for the object to be created.

The CREATE service action was thought to be the appropriate time to establish any object specific attributes, including directing an object to be located near another, locating it with respect to the data rate possibilities on the storage device or establishing any other management policy associated with it.

A possible option (not included yet) is a degree-of-contiguity field. For instance, if a new object should have a minimum degree of contiguity, this attribute could direct the storage device to allocate space in units of that size to ensure that degree of contiguity. Another view holds that this is too "physical" a specification and more appropriate would be a quality of service indicator that specified the performance level required such that the application client describes the requirement rather than the solution.

Some of the object attributes discussed, in addition to those in 7.1, include:

a) Make this file password protected. Rejected as not the right place to put a password.
b) Encrypt this object. This is probably too expensive to put in a disc drive. While it could be done in a subsystem, the security enthusiasts who looked at it thought this was not the right place to do encryption anyhow.
c) Specify if sub object level locking is required. This is still not fleshed out.
d) Specify versioning. This was rejected - as more appropriately done by the OS.
e) Mirror support - cause all updates to be mirrored onto another object. No one has come up with a concrete requirement for this.
f) Allocate space in units of a specified minimum size. The motive for this attribute was to allow the application client to specify a minimum degree of contiguity as mentioned above.
g) Set rights (as in UNIX)
h) Create an object to emulate an SBC storage device. Assign a LUN to it. (LUN returned upon completion.)

B.4 OPEN

Possible options:

Table B.2 — OPEN Option Byte 2 format

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	RDNLY	WRNLY	ATTR	SESS	SEQ

SEQ is set to indicate that this object session should access the object sequentially.

SESS is set to one if the application client requires the OSD device to set up an I/O object session. This may require the OSD device to maintain state for the duration of the open object session. It is made an option because it was thought that many or most I/O would not have quality of service requirements attached to them. In this case no object session state is maintained.

ATTR is set to one to indicate that the information transfer includes attribute data for the object to be created.

WRNLY is set to identify a object session that is to consist of WRITE's only.

RDNLY is set to one, the OSD device is instructed to only allow READ's to the referenced object. WRNLY and RDNLY cannot both be set for a given object session.

There was much discussion on the merits of an OPEN service action. Some felt that OPEN was not needed at all. Some participants felt strongly it should not equate to an application file open. That is, it should not be expected that an OSD OPEN need be issued just because the application submitted a file open to the Operating System. The OSD OPEN provides a point at which quality of service requirements may be expressed to the OSD device. The storage device may, in turn, reject the OPEN if the desired service level cannot be supplied. If no special attributes - such as Quality of Service requirements - were needed, the OPEN may be implicit with the first READ on an object.

One view of the OPEN and CLOSE commands is that they should frame the usage of an object. The storage device may use the awareness of an object not being open as the signal that it may take management action on the object. This might include starting a backup action by informing a backup agent of the candidate object. The OSD device conceivably may also profit from the OPEN and CLOSE by better managing its cache.

Since an important benefit of the OSD is storage management, especially performance, the ability to establish quality of service attributes associated with a particular access of an object is deemed valuable. Thus, if a video

object is to be read sequentially on one occasion for delivery to a customer, the quality of service requirements might be quite a bit more important than if it is being read sequentially simply for backup. It should be possible to express to the OSD device this difference in requirements.

It is thought that a Object_Session_ID will be required to identify under which set of quality of service attributes a given I/O is submitted. For instance a single application client may have both operations underway simultaneously. The OSD device would have no way of knowing for a given read request, whether it had to meet the stringent requirements of the video delivery application or only had to get the data out to satisfy the backup operation. A Object_Session_ID may let the OSD device discriminate between multiple sets of quality of service requests.

If Object_Session_IDs were always used, there would be no need for both a Object_Session_ID and the [User_Object_ID, Object_Group_ID] set on READ, WRITE and APPEND commands. This would simplify the fast path operations, but always require the OSD device to keep state for each OPEN. Since the OSD device's ability to hold OPEN state is finite, it may result in the OSD device being unable to accept an OPEN due to not having space to hold any more state information. Since most operations would not have any QoS requirements, it seems desirable to make the object session an option. This lets the OSD device handle most requests it receives, without having to keep state for all tasks that desire to access an object. This also means that any application client may simply READ an object without first issuing an OPEN (again, assuming no QoS requirements).

The optional length parameter for the OPEN allows an application client to pre-allocate and hold space in anticipation of WRITEs. This is a quality of service feature, persists only for the lifetime of the object session and might be specified with the size attributes in 7.1 rather than an explicit argument on this service action. The application client may cache I/O, with the confidence that there will be space available when the cache is flushed. There might be a better way of doing this; but, clearly, something is needed to support application client caching.

B.5 READ

Possible options:

Table B.3 — READ Option Byte 2 format

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	PRIORITY			Done

It has been argued that a priority capability is redundant to the quality of service attributes, which all agreed would be a more powerful vehicle for managing performance. Still, both are included so that the industry can decide if PRIORITY has a value in the presence of QoS attributes.

There was also a request to have a PRIORITY designation for requests that are not to be executed until a certain amount of idle time has passed. This has not been defined, but could be supported with a set of priority values, such as 12 - 14. A time attribute defining the delay interval would be needed to support this.

A READ may return the entire contents of an object by supplying a length longer than that of the object. The OSD device should return all data and, in the response, the actual length of the data sent to the application client. For instance, a READ with a length of 64K may be used to read any object having a length less than or equal to 64K. The READ should send back as much data as the object contains, with the length of that data supplied in the response.

B.6 WRITE

Possible options:

Table B.4 — WRITE Option Byte 2 format

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	PRIORITY			Done

An attribute on the object to which the WRITE has been issued may cause other events to occur. If mirroring support was called for on the target storage device (as indicated by an attribute on each object), a WRITE may automatically cause the WRITE to be propagated to another OSD to keep it in sync with the written to OSD.

B.7 APPEND

Possible options:

Table B.5 — APPEND Option Byte 2 format

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	PRIORITY			Done

The idea behind the APPEND command is that it leaves to the OSD device the task of concurrency control for a certain class of objects. That is, several application clients may be logging data to an object. If each had to determine the length of the object, acquire an exclusive access rights and write its data, the performance would be far poorer than if each just issues an APPEND. This feature does not work in all cases, of course, but logging and similar operations may be supported where the precise ordering of the log entries is not a concern.

B.8 FLUSH OBJECT

This command may also be used for synchronizing the object group or the storage device by specifying the appropriate value to indicate that one or all OGs are to be flushed.

B.9 CLOSE

The discussions on CLOSE were similar to those on OPEN. Many felt that CLOSE is not needed. Most felt, however, that there was value in identifying to the OSD device that an object was not going to be used by the application client any longer. The OSD device may take action based on this knowledge. One possible action would be to direct an agent to back up the object. This might be desirable if the object had just been updated and had an attribute that called for backing up any time the object was updated. There was stronger support for CLOSE than OPEN.

Any changes as a result of writing to the object, if not already written to the media, may be committed at this time.

It was recognized that an object being created should not be left in an ambiguous state if the CLOSE is not received. The OSD device may either discard the partially created object or it may establish the existence with the data that has been written to it so far. Which of these to actions to take may be a matter of policy, established at the OSD device, object group or object level.

Some believe that application client failure recovery is made harder by requiring a CLOSE in order to not lose written data, especially as the FLUSH operations may be used for ensuring written data is on stable media. An

alternative view of CLOSE is that, independent of zero or multiple proceeding OPEN service actions, a CLOSE service action on an object is an assertion that application client activity with this object has ceased.

B.10 REMOVE OBJECT

Possible options:

Table B.6 — REMOVE OBJECT Option Byte 2 format

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	DESTR

DESTR is set to one to instruct the OSD device that it should obliterate the data in the object to be removed so that no trace is left on the media (i.e., security erase).One issue that needs to be resolved is the issuance of a REMOVE service action when the object is still open as a result of some other activity. The prevailing view was that it should be rejected. This would only work in an environment that consistently issued OPEN and CLOSE service actions.

B.11 CREATE OBJECT GROUP and REMOVE OBJECT GROUP

There was considerable disagreement on the need for OGs, and what form they should take. The great majority of responses was that they were not needed. Others pointed out how an object group concept could be useful, especially in support of legacy OGs because it is quite likely that different file systems, databases, volume managers and virtual memory systems may not be coded to use distributed capacity allocation protocol among themselves. The operation supporting OGs are included not to necessarily endorse the need for them. It is hoped that, first, they will serve to flag the question as to whether they are needed, and, second, to suggest how they might be implemented should OSs be deemed a requirement. The preferred definition was that an object group defined a collection of objects. There would not be a physical segmentation of the storage device tied to the object group. It does not describe a subset of the storage capacity. There may be a capacity quota associated with an object group, which could be used in legacy OSs as a limit on the amount of space consumed by the objects in an object group. This of course leaves open the whole question of over-commitment. There probably should be a choice of policy as to whether the capacity of the storage device may be oversubscribed or not.

This function will also create a well-known object holding object group attributes, which should not be removed as long as the object group exists. The root object may serve as the starting point for navigating the objects in the object group.

The User_Object_ID length was a subject of some discussion. While most felt the longer (i.e. 64 bit) field was appropriate, an argument for efficiency held that 32 bit was sufficient. The outcome was that the field was defined as 64 bits, with the possibility of defining an option bit that would restrict the length to 32 bits.

B.12 IMPORT OBJECT

Possible options:

Table B.7 — IMPORT OBJECT Option Byte 2 format

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	PRIORITY			

It was not unequivocal that IMPORT OBJECT is necessary - or even a very good idea. While the value of moving objects between storage devices without host intervention has value, it was not clear that the oversight of such an

operation should be left to the storage devices. Nevertheless, the command was left in because some saw it usable, especially in smaller system environments.

B.13 GET ATTRIBUTES

GET ATTRIBUTES and SET ATTRIBUTES are used to retrieve or update object characteristics. Originally a 32-bit mask was defined. Some felt that any field mask was just too restrictive at this point. With the prospect of QoS attributes being extremely complex, there probably should be a more open ended mechanism for communicating attribute information between the application client and the OSD device, For example, 5.35.3 suggests a keyword driven mechanism. A 64-bit mask is the implementation described, but this certainly needs more work.

It should be possible to retrieve attributes for multiple objects with a single GET ATTRIBUTE operation. For instance, a single request may return the storage device object (aka root object) and all object group object (aka group object) attributes if there was a convention for describing a list of target objects.

There was some discussion about a list directed operation, whereby the attributes for a set of objects may be retrieved or set. There was no consensus on the format that the list should take. There still needs to be a lot of work on the set attributes an object may have.

B.14 SET ATTRIBUTES

This property is used with special instances of this command to set certain OSD characteristics. The drive key, which is used to manage the OSD enforcement of authentication and security, is set using this command.

The installation-supplied name is also passed to the OSD device by this command. Since many installations may want to ensure that names are not ambiguous, the thought is that setting the name should be a closely controlled operation.

B.15 GET, SET STORAGE DEVICE ASSOCIATIONS

Though not included in the commands, a method for defining and interrogating sets of OSD may be useful, especially in support of RAID configurations.

These service actions may define or interrogate relationships among storage devices. This may be necessary for inter-storage device communications. (A possible implementation may be one of the storage devices being identified as the "master" or first of set, with the others being dependent members of the set. The first of set may be responsible for disseminating to the other members changes in set attributes. Other members may reject attribute settings if they were not from the first of set.)

The motive for defining OSD sets that the storage devices themselves are aware of, is to enable an implementation of a RAID function or mirroring at the OSD level. This corresponds to a software RAID on a string of discs. It was not clear how a RAID function should be provided across OSD without the storage devices knowing the number of storage devices in the RAID group, the arrangement (i.e. order) of the group and the addresses.

Storage device associations, in combination with object based XOR commands make it possible to have controller independent array configurations.

B.16 Object sessions

Object sessions are sets of I/O requests that are to honored by the OSD device in the same way. That is, the application client may use the SET ATTRIBUTE service action to establish the quality of service characteristics associated with the object session. The OSD device is expected to accept the contract - by virtue of not rejecting

the SET ATTRIBUTE service action with SESSION ATTRIBUTES - and process the I/O requests per those quality of service requirements. The Object_Session_ID is returned by the OSD device in the response to an OPEN or CREATE service action. The application client then supplies this Object_Session_ID in each READ, WRITE, or APPEND request. In addition the application client may supply the Object_Session_ID of a previously set up object session in an OPEN to require that the OSD device also process the I/Os for this object with the same quality of service as requested for the object with which the object session was originally established.

B.17 Scatter-Gather operations

To maximize transfer efficiency, scatter-gather variants of the READ and WRITE commands should be considered. The scatter-gather variants should take as arguments a number of {offset, len} tuples describing the sets of bytes to be read/written. The data should be transferred on the wire in the order that it is described in the access. Thus, a READ of { {27,3}, {0,4} } would transfer bytes in the following order: {27, 28, 29, 0, 1, 2, 3}.

The standard should probably describe the atomicity of READ and WRITE operations, as well as the scatter-gather variants. It does not seem necessary to implement "extra" guarantees of atomicity for the scatter-gather variants. That is, "READ-SG { obj 37, { {16384, 8192}, {32768, 8192} } }" does not seem to require atomicity not also provided by performing separately "READ { obj 37, offset 16384, len 8192}" and "READ { obj 37, offset 32768, len 8192}", although implementations of the OSD standard may certainly feel free to provide additional guarantees.

IMPORT OBJECT should be modified to move a range of bytes within an object rather than whole objects. This may be done by adding two offsets and a length to the description of an IMPORT OBJECT operation. One offset would be the offset in the source object, the other the offset in the destination object, and the third is the number of bytes to transfer. A scatter-gather variant, IMPORT-SG, should take a list of such byte ranges to import. Utilizing the scatter-gather variant, a storage manager may perform a restriping operation from an m disk array to an n disk array with at most n*m total IMPORT OBJECT directives without forcing multiple reads or writes of the same bytes.

B.18 Attributes

The distinction between OBJECT LOGICAL LENGTH and OBJECT SIZE is that the former defines the range of addresses in which READ service actions return data and the second reports the real capacity consumed by representing the object's data. Some believe that SET ATTRIBUTES should take a new value of OBJECT LOGICAL LENGTH which, if smaller than the current value, truncates the object (destroying data with addresses beyond the new value) and if larger than the current value, extends the range of addresses that responds to reads, implicitly defining the value of newly addressable addresses to be zeros.

The attribute, DATA_ACCESS_TIME, was discussed extensively. Many felt that the performance cost for maintaining this was excessive, therefore not to be included. Others held that it is necessary to the OSD device intelligently participating in HSM functions.

There was some sentiment for the interpretation of the LAST_ACCESS_TIME being the time of the last OPEN service action on the object.

B.19 Policy

Policy refers to the set of conditions and subsequent actions (to be performed in the event the associated condition is met) connected to the management of data and/or storage. Policies are typically time- or event-driven, are independent of storage geometry, and frequently occur independently of any application processes. Common examples of policies include conditional or time-based backup, archive, delete processing, data movement, and storage device maintenance.

Policies, though relevant to the management and disposition of objects and their contents, are beyond the scope of this standard.

Some felt that even though eliminating a sector dependency was valuable, there still would be a need for a block size attribute to give the application client's good guidance for laying out objects and transferring on wise boundaries. This may eliminate or at least minimize the storage device's need to do read-modify-write sequences due to mis-aligned transfers. Blocksize here refers to the modulus that will align transfers to desirable boundaries. Every object is assumed to begin on such a boundary.

B.20 Rational and justification

The use of attributes to characterize OSD objects stems from the desire to reduce device dependencies within accessing applications, achieve a higher (i.e. simpler and more application pertinent) level of specification, and enable more powerful asynchronous or semi-autonomous functions at the device. In order to achieve these goals, we have examined similar concepts embodied in system-managed storage, QoS, and policy management, and applied them to the object-based device environment. Accordingly, it is felt that this design supports the desired objectives:

a) Reduced device dependencies - the attributes contain no device specifics. An OSD device is free to self-manage and optimize itself as long as the result continues to achieve the attribute specifications;
b) Higher level of specification - the attributes are biased in the direction of processing and/or data requirements, rather than device capabilities; and
c) Function enablement - functions such as data movement, querying, and performance optimization that are often performed at the host may be performed elsewhere in the network (e.g., by a storage management processor) or by the OSD device. Additionally, the extended attribute capability may be used to enable more powerful functions (e.g., data base assists, cryptographics, format translation) to be performed outboard of the host.

Annex C
(Informative)

OSD Related Topics

C.1 Relationship to file systems

Data is stored in fully self-defined and self-contained objects that the storage device is responsible for maintaining. Within the constraints of the security policy, the object-based architecture makes it possible for any application client to inspect a storage device to discover what objects exist there and access them. The application client need not know anything about the OSD device's physical characteristics or even the operating system that created the objects.

While objects might appear to look and work very much like traditional files, they are not the same. There is no hierarchical organization as in most file systems. Although it may be that the OSD-object-group-object organization is, in fact, a hierarchy. There is no naming function. An object may contain several files, or an object may only be part of a file. It is assumed that Objects may be operating system independent, with almost any OS able to super-impose its file system structures on the object abstraction. The OS files, directories and attributes, other than space management mechanisms, are constructed as objects.

The object abstraction makes available information valuable to the optimal exploitation of the storage. For quality of service and management reasons the object semantics should expose enough of the storage device's capabilities without infringing on its self-management. Capabilities are not meant here to be a description of hardware, but rather, performance characteristics, such as the data rate, time to first byte, and other attributes described above. For instance, a typical disc drive today has several strata of transfer rates. The specifics of these should be known to a degree sufficient to allow a file allocation policy to take advantage of them. This may take the form of service levels and capacity quotas for each.

Reliability characteristics are also be attributes. A disc array may include mirror, RAID and JBOD storage. An object may be created on the type of storage that provides the appropriate reliability level as a result of a reliability attribute supplied with a CREATE action.

C.2 Object groups

Operating systems commonly provide for allocating disc space into one or more mutually exclusive regions, called partitions. A similar facility is available on an OSD device with this significant difference: the OSD object groups are not divisions of the storage space, but simply logical collections of objects.

One use of the object groups is to segregate the name space of an OSD device so that multiple, non-cooperating managers (such as file systems, VM pagers, LVMs, or database managers) may use the same OSD without conflict. There may optionally be a capacity quota associated with an object group to enable management of space consumption by the objects in the object group. Each object group may have a capacity quota associated with it for this purpose. This may be used to protect against objects consuming space unreasonably. Should an object group fill, and there still be available space on the storage device, space may be added to the object group quota by subtracting from another object group quota.

Not tying object groups to specific amounts or segments of the storage device capacity has another benefit. Every disc drive and many disc arrays have zones offering different data rates. For some applications, such as video or image processing, it may be important to be able to put certain files in the high data rate zones of the storage. If object groups actually divided up the space, it might be that only the first object group could hold objects that would have the property of the highest data rate a storage device offered. By not tying object groups to storage location,

multiple object groups may contain objects in the high data rate zones (by specifying a quality of service attribute that called for such allocation).

This makes it possible for objects to be grouped into object groups for management purposes, with each of the object groups able to offer high data rate allocation, up to the capacity limit of those zones. A video processing shop might have several projects underway simultaneously. The objects for each project may be stored in a separate object group with the video images being allocated in the outer zones and the program files or control information being put in lower data rate zones.

The storage device maintains an object list for each of its object groups and space management information for the entire storage device.

C.3 Identifiers (IDs)

The identifier associated with an object is chosen by the storage device and returned to the application client in response to the command to create an object. The ID will be an unsigned 64-bit integer. Specific ID's may be reserved for well-known objects. Well known objects are needed to enable a file system to start navigating through the objects on the OSD device.

Allowing the application client to supply OBJECT ID at CREATE time, instead of having the OSD device assign and return it may have made garbage collection easier for the application client, but would have introduced a severe performance penalty on CREATE. The OSD device would have to verify that the OBJECT ID was not already in use, which could take a very long time on a storage device that had hundreds of thousand of Objects. So, it has been specified that the OSD device assigns it.

C.4 Relationship to Sector Based Storage devices

C.4.1 Emulating an SBC device on an OSD device

While there will likely be a continuing need to support SBC and relative sector addressing, there is a security problem with allowing a storage device to switch from OSD to SBC and back. To ensure data security the change from SBC to OSD and the change from OSD to SBC should obliterate the contents of the storage device. The mechanism to accomplish the change is vendor specific.

It should be possible to emulate a SBC storage device on an OSD device. The OSD device may allocate an object matching the desired size of the logical SBC. It assigns this a LUN, letting an application client that cannot support the OSA build its file system and access data in this special object as though it were a separate physical storage device.

Editors Note 25 - GEM: This capability may not survive the development.

C.4.2 SBC SCSI commands

The response to the following SCSI commands would be invalid operations in OSD mode:

FORMAT UNIT	READ LONG	SEEK	WRITE LONG
PRE-FETCH	REASSIGN BLOCKS	SEEK EXTENDED	WRITE SAME
READ	REBUILD	VERIFY	XDREAD
READ CAPACITY	REGENERATE	WRITE	XDWRITE
READ DEFECT	RELEASE	WRITE AND VERIFY	XPWRITE
READ EXTENDED	RESERVE	WRITE EXTENDED\	

C.5 Data sharing and concurrent update

Scalability may be improved with an optimistic control scheme where application clients may act as though there is no conflict unless there are actually simultaneous attempts to access the same records by multiple application clients. They may learn this from the OSD device when attempting to gain control of objects for the purpose of updating them. An attribute set by an application client establishes that it intends to update certain data. The attribute is reset after completion of the WRITEs. Only if another application client attempts to set the same attribute is there a need to resolve a conflict. This approach seems to have the potential to greatly reduce the inter-system traffic servers generate to maintain data consistency.

A flexible but simple extension to the action set of clause 6 to support concurrency control at the OSD device is to make some actions conditional on attribute values and allow successful conditional actions to "set" attributes atomically. For example the attribute modifier structure of 5.2.1.1 may be applied to OPEN, READ, WRITE, APPEND and SET ATTRIBUTE actions before these actions make any change to an OSD state or transmit any data. Specifically, if each of a sequence of attribute modifiers evaluates successfully before the rest of the action is enacted, then attribute values may be tested and modified and action may be conditional. In this scenario, the semantics of 5.2.1.1 should be extended with a byte offset and bit length (because some attributes, such as the FS-specific field, are large enough to be unwieldy to manipulate as a primitive value) and the Comparator values should be extended to differentiate between "test current value" and "overwrite current value" (which always succeeds).

C.6 OSD and aggregation

C.6.1 Overview

Aggregation describes data structures that span storage devices. Three common uses are redundancy, performance and capacity. Traditional storage architectures implement these on a block level interface. An OSD device hides the block specifics, so the equivalent functions are made available at the object interface of the OSD device.

In the case of capacity spanning, for instance, there is often a need to have files allocated across storage devices. A large database might span several drives. A file system may create, access and manage a collection of objects located on several storage devices as if they constituted a single file.

When an application client solicits permission to access a file that is held in multiple objects, it should also have mapping information that will let the application client direct its I/O commands to the appropriate storage device. It is transparent to OSD devices and the objects they contain that they may be only parts of larger storage or data constructs. (If defined, Storage device Association is an exception.) This does not preclude an OSD device subsystem from internally aggregating among OSD logical units it manages.

C.6.2 Aggregation for redundancy — RAID and mirroring

Mirroring and RAID are used to protect against the loss of a storage device by making it possible to recover the data located on the lost storage device using data stored on other storage devices. The XOR functionality may be adapted to work on the object interface. XOR operations may be done implicitly whenever a WRITE command addresses an OSD device that is parity protected. The OSD device may cause the additional steps necessary to maintain the parity protection to take place by converting the WRITE to the equivalent of an XDWRITE, either standard or third party version. It may cause the following:

1) The old data to be read
2) The new data to be written
3) The two above to be XOR'd together
4) The target OSD on which the parity is to be updated to be calculated
5) Both #3 and #4 above to be returned to the application client.

The application client could then issue a WRITE to the parity object on the destination storage device, resulting in the equivalent of an XPWRITE. Note that this is a possible exception to the rule of OSD not being cognizant of other OSD. This implementation of RAID support is more efficient if each storage device in the parity set being aware of its membership in the set including the identity of each other member.

In an alternative method, the OSD device could issue a third party WRITE command to the appropriate parity drive to complete the XPWRITE part of the parity protection.

Mirroring can be handled in at least two ways. The application client can be responsible for ensuring the WRITEs are issued to both storage devices, or the mirror storage devices may take responsibility for propagating WRITEs to the other. Determining which of these is the more desirable may depend on the needs of a particular installation, but both may be used as each method has its advantage.

C.6.3 Aggregation for capacity — spanning

As described above, for databases and other large files that do not fit on a single storage device, there is a method for spreading them across several devices. This has been done in legacy systems by creating logical volumes that span disc drives. When an application client needs to access such a database, the volume manager is responsible for directing the request to the appropriate drive, based on the displacement of the request into the data. The OSA uses a similar method.

The application client needs a mapping function similar to that of the volume manager in a SBC environment. When the application client accesses the Policy/Storage Manager to get authorization to access the database, the spanning information is also returned. The application client may use this to transform a file request into an object request to the appropriate OSD.

C.6.4 Aggregation for performance — striping

Striping is handled in the same way as spanning, except that a single large-data request results in an object request to each of the storage devices in the stripe group.

C.6.5 Accessing aggregated objects

The following is a description of one method for accessing aggregated objects. It assumes an option in Option Byte 1:

Table C.1 — Option Byte 1 format supporting aggregation

Bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	DPO	FUA	Reserved	Reserved	NO-REDIR

NO-REDIR set to one indicates that the application client does not support mapping of an object onto multiple OSD and does not accept mapping information.

When a requestor accesses an object, it first obtains a valid capability (see C.2.3) for doing so. One way that the requestor might accomplish this is that it might possess the necessary keys to compute a valid capability for accessing the given object on the given OSD device. Alternatively it may negotiate with an external service (such as a file system policy manager) to obtain this information. The ways in which an application client might perform such a negotiation are beyond the scope of this standard. The essence of this suggested method for accessing aggregated objects is that an application client discover the nature of an aggregated object (that is, its mapping) as a side effect of trying to access the aggregated object as if it was a simple (single OSD) object. An aggregated object that is being accessed as a simple object is referred to as a virtual object and the OSD device that is named in accessing a virtual object is referred to as a Storage Manager (which may actually be a Policy/Storage Manager). The term capability in this context refers to a security token described in B.1. For the purposes of this section, a capability may be considered an opaque set of bytes provided for security purposes. However, because a privileged key may be included in a capability, it is recommended that Storage Managers encrypt responses that include maps.

Editors Note 26 - ROW: Subclause C.2.3 does not exist in OSD revision 3, so there is no way to determine the correct replacement cross reference. Subclause B.1 does not describe a security token to the best of my knowledge. The cross reference was once to C.6.4, but that appears to have changed a revision or two ago.

The Storage Manager named by a virtual object shall respond to requests as any OSD would. However, to realize the scaling benefits possible with network-based storage, it is desirable that application clients be able to directly access individual OSD devices providing backing store for virtual (aggregated) objects. To do this, application client need to become aware that the object they are addressing as {OSD-NAME, Object_Group_ID, User_Object_ID} is in fact mapped over for example {{OSD-NAME1, Object_Group_ID1, User_Object_ID1}, {OSD-NAME2, Object_Group_ID2, User_Object_ID2}, …}.

In responding to a virtual object request, the Storage Manager may choose to include in its response the mapping of the virtual object onto member OSD devices. It may also choose to act as a proxy on behalf of the requestor. If the NO-REDIR bit is set in Option Byte 1 of the request, the Storage Manager acts as a proxy on behalf of the requestor, and may not include the mapping of the object onto the aggregate members.

C.6.6 Description of aggregate layouts

To ensure the ability of all application clients to be able to at least throw away layout descriptions that they are not able to parse, layout descriptions begin with the number of bytes used to describe the following layout (excluding the length field). After the layout descriptor length field (4 bytes) is the first byte of the layout, which is a layout type field (1 byte). If the layout type is not recognized, the application client may use the descriptor length field to

discard the entire layout description, and by remembering to set the no-redirect bit for future accesses to that virtual object, rely on the Storage Manager to proxy all accesses on that virtual object.

The following description uses these abstract data types:

a) An Obj-Descrip is a tuple containing {OSD-NAME, Object_Group_ID, User_Object_ID, SIGNED CAPABILITY}, where signed capabilities are described in C.2.3
b) A fully-qualified layout description is a tuple containing {layout descriptor length, layout type, type-specific layout description} where a type-specific layout description may contain one or more embedded fully-qualified layout descriptions.

Editors Note 27 - ROW: Subclause C.2.3 does not exist in OSD revision 3, so there is no way to determine the correct replacement cross reference.

The following are defined for type-specific layout descriptions:

a) Simple mirrored set (type=1). The first 16 bits of the type-specific layout description represent the number of elements in the mirrored set, and the remaining bits contain a list of the length specified by the number of elements of Obj-Descrips. Note that this may also be used to represent the storage management option of a non-mirrored, non-striped object by describing a mirrored set with only one member.
b) Mirrored set (type=2). The first 16 bits of the type-specific layout description represent the number of elements in the mirrored set. The remaining bits contain a list, of length as specified in the preceding field, of fully-qualified layout descriptions. This layout type differs from a simple mirrored set in that where the simple mirrored layout is a set of Objects, the mirrored set allows the elements of its set to be virtual objects, each may have different layouts.

Editors Note 28 - ROW: So a mirrored is an object group? Having given the term 'object' a very narrow meaning, you must now be careful not to use it in a broad meaning.

---

c) Striped set (type=3). The first 16 bits are a count of the number of (virtual) objects that the described object's data is striped over. The next 32 bits are the stripe unit size (the number of contiguous bytes in the described object mapped to one constituent object). The remaining bits contain a list, of length given by the first field, of fully-qualified layout descriptions. The order that data in the described object is striped over constituent objects is the order of constituent fully-qualified layout descriptions in this list.
d) RAID-5 left symmetric set (type=4). The first 16 bits contain the number of stripe units in the parity-protected "group". That is, the first field's value is one more than a count of the number of (virtual) objects that the described object's data is striped over. The next 32 bits are the stripe unit size (the number of contiguous data bytes in the described object mapped to one constituent object). In each parity-protected group, one constituent object stores a bit-wise parity of the stripe units of the rest of the group. The assignment of parity to constituent objects follows from the RAID level 5 parity rotation called left-symmetric. The rest of the bits contain a list, of length specified by the first field, of fully-qualified layout descriptions.
e) Concatenated set (type=5). The first 16 bits are a count of the number of objects that are used to contain the data. The remaining bits are a list, of length given by the first field, of fully-qualified layout descriptions of constituent objects which, if concatenated in the given order, represent the content of the described virtual object. The constituent objects may have arbitrary sizes individually, requiring a application client to obtain attributes of each constituent object before issuing re-directed object accesses.
f) Vendor-specific set (type=6). The first 32 bits of the description are a vendor id, followed by a 16-bit value representing a vendor-specific layout type and the (arbitrarily long) vendor-type-specific layout arguments.

This allows storage system vendors to sell both Storage Manager and application client software with specialized layout algorithms.

NOTE 9 Most layout description types may be nested. This allows the description of layouts such as "mirrored, striped" without rapidly consuming the layout type namespace. The "Simple mirrored set" type is the only one defined to disallow nesting. Most layout descriptions will utilize this as the root type to describe the layout. Individual vendors might define non-nested layout types, although this is not recommended.

C.6.7 Other type values

Other layout type values that may be considered for standardization: RAID level 6 (XOR parity and Reed-Solomon parity); EvenOdd; non-rotating XOR parity (that is, RAID level 4); parity declustering.

C.6.8 Example 1: Simple mirrored pair

In this example, a virtual object is mapped on a pair of constituent objects on two different OSD devices. Each object is a mirror of the other. Capabilities and their size (capability-sz) are discussed in C.2.3.

Table C.2 — Aggregation: Simple Mirroring Descriptor

Value	Size (in bytes)	Description
3 + 2 *(16+4+8+ capability-sz)	4	size of subsequent layout description
1	1	layout type (1=simple mirrored set)
2	2	Number of objects in mirrored set
Obj-Descrip-1	16+4+8+capability-sz	Descriptor for a non-virtual object
Obj-Descrip-2	16+4+8+capability-sz	Descriptor for a non-virtual object

Editors Note 29 - ROW: Subclause C.2.3 does not exist in OSD revision 3, so there is no way to determine the correct replacement cross reference.

C.6.9 Example 2: Simple striping

In this example, a virtual object is striped over four simple objects with a stripe unit size of 4 KB. For convenience the following refers to the size of an Obj-Descrip as obj-descrip-sz (which, by the preceding example, is 16+4+8+capability-sz bytes).

Table C.3 — Aggregation: Striping Descriptor (part 1 of 2)

Value	Size (in bytes)	Description
7 + 4 * (4+3+obj-descrip-sz)	4	size of subsequent layout description
3	1	layout type (3=striped)
4	2	count of objects in striped set
4096	4	stripe unit size, in bytes, for striped set

Table C.3 — Aggregation: Striping Descriptor (part 2 of 2)

Value	Size (in bytes)	Description
obj-descrip-sz+3	4	size of layout description of stripe member 1
1	1	layout type (1=simple mirrored set)
1	2	number of items in mirrored set
obj-descrip1	obj-descrip-sz	object 1 description (OSD-name, Group. Object, Capability)
obj-descrip-sz+3	4	size of layout description of stripe member 2
1	1	layout type (1=simple mirrored set)
1	2	number of items in mirrored set
obj-descrip2	obj-descrip-sz	object 2 description (OSD-name, Group. Object, Capability)
obj-descrip-sz+3	4	size of layout description of stripe member 3
1	1	layout type (1=simple mirrored set)
1	2	number of items in mirrored set
obj-descrip3	obj-descrip-sz	object 3 description (OSD-name, Group. Object, Capability)
obj-descrip-sz+3	4	size of layout description of stripe member 4
1	1	layout type (1=simple mirrored set)
1	2	number of items in mirrored set
obj-descrip4	obj-descrip-sz	object 4 description (OSD-name, Group. Object, Capability)

C.6.10 Storage managers responding to requests

There are four possibilities for what an application client might find included in the response from a Storage Manager: If both a map and a command response are included, the response is sent first. Hence, an application

client allocates at least 4 bytes more than the expected response size to be able to determine if a valid map was received (because an application client does not necessarily know the size of the included map).

Table C.4 — Aggregation: Responses

Response Type	MAP Included	RESPONSE Included	Description
1	Y	Y	The Storage Manager completes the request on behalf of the requestor, and forwards the result back. Additionally, the mapping of the object onto the aggregate members is included in the response. This response is disallowed when NO-REDIR is specified in the request.
2	Y	N	The Storage Manager responds with only the mapping of the object onto the aggregate members. The application client repeats the request, either by mapping the request onto the members of the aggregate, or back to the Storage Manager after setting the NO-REDIR bit in Option Byte 1. This should be disallowed when NO-REDIR is specified.
3	N	Y	The Storage Manager transparently completes the request on behalf of the application client, and forwards the result to the application client. To the application client, it appears as if the object named by Obj-Descrip is merely a single simple object on a single OSD.
4	N	N	The Storage Manager does not complete the request, and no mapping for the object is provided to the Requestor. This result is considered an error, and as such, the Storage Manager shall provide an error code indicating why the request was not completed (for example, an argument was invalid).

When NO-REDIR is specified for a request, response type=3 is the only valid successful result of an operation.

C.6.11 Example 1

In this example, an application client already holds a valid capability for performing GET ATTRIBUTES and READ operations on a virtual (aggregated) object (V_obj_id). The capability was provided by a separate Policy/Storage Manager that may not know the mapping for the virtual object. The Storage Manager that backs the virtual object is assumed to have in its cache the attributes of the virtual object and knows that the virtual object is striped over simple objects on OSD1, OSD2, and OSD3. In the example, the application client acquires the virtual object's

attributes, providing sufficient space to receive the map, then reads at least three times the virtual object's stripe unit size.

Table C.5 — Example 1: Read Aggregated Object

Step	Service Action	Key Attributes	Source	Destination	Options
1	GET ATTRIBUTE	V_obj_id	Application Client	Storage Manager	-
2	GET ATTRIBUTE response	Attributes, map	Storage Manager	Application Client	Response, map included
3	READ	obj_id-1	Application Client	OSD1	-
4	READ	obj_id-2	Application Client	OSD2	-
5	READ	obj_id-3	Application Client	OSD3	-
6	READ response	data	OSD1	Application Client	-
7	READ response	data	OSD2	Application Client	-
8	READ response	data	OSD3	Application Client	-

If the requestor did not recognize the map provided in step 2, or is not able to implement the layout specified by the map, the result might be:

Table C.6 — Example 1: Read Aggregated Object without mapping support

Step	Service Action	Key Attributes	Source	Destination	Options
1	GET ATTRIBUTE	V_obj_id	Application Client	Storage Manager	-
2	GET ATTRIBUTE response	Attributes, map	Storage Manager	Application Client	Response, map included
3	READ	V_obj_id	Application Client	Storage Manager	NO-REDIR
4	READ response	data	Storage Manager	Application Client	-

C.6.12 Example 2

This example is identical to the previous, except that the requestor does not perform a GET ATTRIBUTES operation before performing the READ and the Storage Manager does not provide any data with the map it returns on the first read of the virtual object.

Table C.7 — Example 2: Read Aggregated Object

Step	Service Action	Key Attributes	Source	Destination	Options
1	READ	V_obj_id	Application Client	Storage Manager	-
2	READ response	map	Storage Manager	Application Client	map included
3	READ	obj_id-1	Application Client	OSD1	-
4	READ	obj_id-2	Application Client	OSD2	-
5	READ	obj_id-3	Application Client	OSD3	-
6	READ response	data	OSD1	Application Client	-
7	READ response	data	OSD2	Application Client	-
8	READ response	data	OSD3	Application Client	-

Editors Note 30 - ROW: Revision 3 contains what appears to be a second copy of table C.7 at the top of

the page immediately following the table. The duplicate table has not be reproduced in this revision.

In this example, if the application client does not perform the mapping, the results are:

Table C.8 — Example 2: Read Aggregated Object without mapping support

Step	Service Action	Key Attributes	Source	Destination	Options
1	READ	V_obj_id	Application Client	Storage Manager	-
2	READ response	map	Storage Manager	Application Client	map included
3	READ	V_obj_id	Application Client	Storage Manager	NO-REDIR
4	READ response	data	Storage Manager	Application Client	-

C.7 Booting from an OSD device

C.7.1 Cold boot

This sequence should not be much different from a sequence on SBC. The same files need to be located and read. The page or swap files have the same function and be used in the same way. To find "boot blocks" and other bootstrapping data a boot EPROM uses attributes on well-known objects such as an object group group object, instead of fixed sector addresses used by SBCs.

C.7.2 Warm boot

The warm boot process exposes a fundamental conflict between the desires of minimized boot up time versus full security. OSD may provide several choices on how to achieve balance among the choices.

During the shutdown of some systems, it may be fairly easy for a system to define a quick boot image object and write the required state to it. This may be a well-known object or one defined for this purpose and named as an attribute to a well-known object. Alternatively, many systems have the ability to keep a small amount of state information, such as a boot object ID, through shutdowns. It would only require a very small amount of BIOS code to OPEN and READ this object. No directory would have to be searched, no location information would have to be kept. The system may read this object to recover its running state and be up and going quickly. The application client writing the boot image may embed in the image a security code (signature) that only it could decipher. This would let the system identify a boot image that has been corrupted.

Another possibility is that the SAN discovery process may reveal, to the system starting up, the list of boot storage devices in some priority order. The low level code in the system may go down this list to find a valid, well-known boot object. The access control on this object may be a special case allowing READs without the need for authentication. It is conceivable that a denial of service attack may be used on this object. Since anyone can read it with no special permission, an intruder may continuously read to disrupt the other systems that have a valid requirement to do so. Note: The object abstraction offers several advantages here. There is no need to keep track of any physical disc location. In fact even if the Boot Object were moved on the storage device, the system may just as readily read it. Also the recorded state may include several open files. The object attributes may identify if any of these were changed or invalidated since the shutdown, making it possible to either quickly come up or to identify right away that the state information is outdated and a cold boot should be done.

C.8 External dependencies

OSD is an enabling technology. OSD provides robust, platform-independent access to storage objects that are designed so that almost any file system may be built using their capabilities. However, other I/O subsystem compo-

nents need to evolve to take advantage of OSD properties in order for those benefits to be realized. The following table summarizes the evolution of other system components necessary in order for consumers to realize the benefits of OSD.

Table C.9 — OSD Dependencies

Component	What is required
File system and other data managers	File systems typically manage their own storage capacity. To effectively use the capabilities of an OSD device, a file system would have to be rewritten to use the OSD device's storage management functions, while retaining its own naming conventions, directory structure, and access semantics. A further possibility arises when OSD is combined with user mode network access, as with the Virtual Interface Architecture (VI). VI allows file systems and database managers to communicate directly with OSD devices without traversing the system I/O or network stack once a connection is established. The low latency achieved thereby enables clusters of computers to share access to storage resources (for "shared nothing" clusters) or to share data with minimal inter-node locking traffic. Exploitation of this feature requires modification of the data managers to use a VI interface, in addition to the modifications listed above.
I/O software stack	OSD devices require commands and responses that are not part of legacy storage device driver stacks. Drivers have to evolve to issue OSD commands and respond appropriately. APIs have to be developed to allow file systems, database managers and applications to issue these commands.
Cluster software	An OSD device is aware of both the data constituting a data Object and the attributes of data Objects. It may therefore perform certain management functions (e.g., defragmentation, shuffling for performance optimization) autonomously, and in a host-independent way.
Management tools	Depending upon the extent to which management features are implemented in OSD devices, data management tools (e.g., backup managers and hierarchical storage management products) have to be modified to take advantage of those features. In general, the change required for management tools to utilize OSD capabilities is a change from being the data movement engine to being the director of the OSD device's data movement engine.

Annex D
(Informative)

Known Unresolved Issues or Uncompleted Topics

D.1 Audit Trails

An audit trail is a history of all instances of all (or of a subset of) operations applied to the OSD device. Audit trails are used by security analysis and performance management tools. However, the necessity of logging an audit trail of OSA activity to OSD devices may pose capacity, throughput and fault tolerance problems. These may be ameliorated by having the option to log the audit trail on another network storage device dedicated to that function or by including in the OSD device temporary storage to delay and group audit records before transfer to OSD device media. Even so, the bandwidth consumed by a sizable number of storage devices logging records would reduce the application bandwidth.

In a transaction environment a single extra message for each I/O significantly degrades performance. All I/O is essentially a message and the I/O subsystem is typically message bound.

D.2 Clocks

Each storage device should have a readable monotonically incrementing clock to be used for time stamping secure messages and object attributes. Either this clock should be synchronized securely system wide, or the server will need to accommodate the discrepancies in values from storage device to storage device. The system will need to provide a mechanism to reload the clock, or the server's accommodation, should the storage device lose power.

Time stamps on object attributes may not have the accuracy that the same stamps would have if they were constructed by a server-based Policy/Storage Manager software or application clients. Such server systems may be accommodated by OSA because the server software may construct their own time stamps and record them in the FS-specific attribute space (uninterpreted by the OSD device). However, very inaccurate (low resolution) OSD clocks still have significant impact on timestamp-based security protocols (detection and elimination of overheard and replayed commands is often based on message timestamps).

The representation of time should perhaps be larger. 32 bits will be inadequate for a fine-grained timer that records elapsed time since some canonical epoch (such as midnight Jan 1 1970 GMT) and is required to never wrap around. Also, the definition of the clock as a counter fails to specify a resolution, which allows implementations that increment the clock in a non-temporal manner (for example, if every arriving message increments the clock, it is monotonically non-decreasing). One recommendation calls for a 64-bit clock that represents nanoseconds elapsed since the canonical epoch. This allows enough resolution to bind the value represented by this clock to elapsed time, and still provide uniqueness of timestamps without fear of wrapping (that clock would not wrap until June of AD 2554). Another common 64-bit representation of time utilizes the high 32-bits for the number of seconds from the epoch, and the low 32-bits for the number of nanoseconds since the last second tick. This suffers from three problems. One is that the 32-bit second clock will wrap in 2038. The other is that a large number of values that this 64-bit number may contain are invalid, necessitating extra sanity checks. The third is that simple arithmetic and comparisons involving these values requires additional complexity, whereas comparing, adding, and subtracting 64-bit integers is comparatively simple.

D.3 List Directed Operations

Almost all file systems offer the ability to get the attributes of a set of files and, often, to change the attributes on a set. There is not yet provision for a similar capability on OSA objects. It would not be difficult to define, but it is thought that file systems requirements need to be taken into consideration in determining the most appropriate approach. It will need to include a means for getting the status of all object sessions by Object_Session_ID.

D.4 Responses

A single byte has been allocated to hold the standard SCSI response. But the OSD device may have need of more error reporting capability than is possible in a single byte. Addressing

The limits on node addressing may be too restrictive. In addition, the fixed association of application client with a single network address is too restricting. An host system may easily have two or more NICs. It should be possible to take advantage of this without explicit direction from the application client to change from using one address to using another.

Editors Note 31 - GEM: Why isn't this transparent to OSD?

D.5 Security

D.5.1 SET KEY operation

Since it is desirable to change the keys in the key hierarchy on a regular basis to avoid key compromise, it is necessary to provide some mechanism for changing keys. SET KEY should be added for this purpose.

The SET KEY operation has a minimum security requirement of all 1s (i.e., encryption and authentication are mandatory). The SET KEY should be authorized by a null-flavored capability signed with a key that is at the same level or higher in the key hierarchy.

Because the master key is the first key to be set on the drive, and its secrecy is the most critical, it is recommended that alteration of the master key only be performed when the drive is attached to a small, secure network. Loss of the master key should be considered a catastrophic failure and the device destroyed, security erased, or reformatted as determined by the policy manager.

It is recommended that the lower-level keys in the hierarchy be changed more frequently than the upper-level keys, and that the lowest-level key possible be used for routine operations. This avoids unnecessary exposure of high-level keys.

Annex E
(Informative)

Motivation for the NSIC OSD

E.1 Overview

NSIC uses the term NASD to mean *Network Attached Storage device*. In this context, a *storage device* is anything that provides persistent storage and represents itself to hosts as a single entity for the purpose of transmitting or receiving commands and data. By a strict interpretation of this definition, any storage device that attaches directly to a network could be called a NASD. In practice, however, the NSIC/NASD Working Group used the term to denote storage devices whose general character is similar to the Carnegie Mellon University Parallel Data Laboratory's ***Network Attached Secure Disk***. See references in the bibliography.

The NSIC/NASD Working Group chose a subset of the full potential functionality of NASD for its first standardization effort. This subset is the Object Based Storage Device (OSD) described in this standard.

E.2 Potential OSD products

The NSIC/NASD Working Group defined an architecture for object-based storage devices. The definition deliberately encompassed the concept of *virtual* OSD devices exported by storage aggregators (e.g., RAID controllers) as well as actual storage devices (e.g., disc drives, tape drives, and solid state discs). The first OSD "storage devices" introduced to the market may be virtual storage devices instantiated by controllers managing conventional discs. While sacrificing the fine scaling granularity that would result from physical OSD devices, the controller approach may have the advantage of deferring the necessity to implement host- or Policy/Storage Manager-assisted aggregation in order to gain OSD benefits.

E.3 Benefits of OSD

The NSIC/NASD Working Group was motivated to develop an architecture for OSD devices by the expectation that OSD devices will deliver value to consumers of storage subsystems. The properties are believed to be particularly attractive for clustered computing. The expected benefits are as follows:

a) higher quality storage management operations with less host effort;
b) less human intervention;
c) tighter and more timely control;
d) controlled sharing of data;
e) easier heterogeneous computing;
f) minimize data access synchronization requirements for clustered servers; and
g) performance benefits in a clustered system architecture.

The attributes never leaves the storage device, eliminating a certain amount of I/O.

The storage device may know which objects are closed or open, and is able to use that information to more effectively manage its cache.

Prefetching may be much more effective as the storage device knows the layout of the object being read. The storage device may more effectively determine sequential access patterns.

The cache in the storage device may hold attributes once for multiple systems accessing it.

The storage device may participate in quality of service decisions, such as where to locate data most appropriately.

Adding a storage device also adds an engine to manage its space. This should contribute to better scalability by not burdening a server with additional processing requirements for each storage device attached.

Annex F
(Informative)

Bibliography

Borowsky, E., et al, "Eliminating storage headaches through self-management", www.hpl.hp.com/SSP/papers/IWQoS97.pdf

Golding, R et al, "Attribute-managed Storage", October 1995. www.hpl.hp.com/SSP/papers/MSIO.pdf

Bellare, M., et. al., "Keying Hash Functions for Message Authentication", Crypto '96, 1996.

Gibson, G, et al, "File systems for Network-Attached Secure Disks", March 1997. www.pdl.cs.cmu.edu/PDL-FTP/NASD/CMU-CS-97-118.pdf

Gibson, G., et al., "File Server Scaling with Network-Attached Secure Disks", ACM SIGMETRICS, June 1997. www.pdl.cs.cmu.edu/PDL-FTP/NASD/Sigmetrics97.pdf

Gobioff, H., Gibson, G., Tygar, D., "Security for Network Attached Storage Devices", CMU-CS-97-185, October 1997. www.pdl.cs.cmu.edu/PDL-FTP/NASD/CMU-CS-97-185.pdf

Gibson, G et al. "A Cost Effective, High-Bandwidth Storage Architecture," 8th ASPLOS, October 1998. www.pdl.cs.cmu.edu/PDL-FTP/NASD/asplos98.pdf

Amiri, K., Gibson, G, Golding, R., "Scalable Concurrency Control and Recovery for Shared Storage Arrays, CMU-CS-99-111, February 1999. www.pdl.cs.cmu.edu/PDL-FTP/NASD/CMU-CS-99-111.pdf