T10

Working draft NCITS TBD-200X Project 1355D

Revision 3
1 October 2000

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

Abstract: This standard specifies the functional requirements for the Object Based Storage commands. 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.

T10 Technical Editor:

Gene Milligan
Seagate Technology Inc. OKM151
10321 West Reno
Oklahoma City, OK 73127-9705
P.O. Box 12313
Oklahoma City, OK 73157-2313
USA

Tel: (405) 324-3070
Fax: (405) 324-3794
Email: gene_milligan@notes.seagate.com

Reference number
ISO/IEC ***** : 199x
ANSI NCITS 306 - 199x
Printed October, 2, 2000 7:45AM~~October, 1, 2000 7:42PM~~

Other 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) 533-7036
Email: lohmeyer@ix.netcom.com

T10 Vice-Chair
George O. Penokie
IBM
Department 2B7
3605 Highway 52 North
Rochester, MN 55901
Tel: (507) 253-5208
Fax: (507) 253-2880
Email: gop@us.ibm.com

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

T10 Web Site www.t10.org

T10 Reflector: To subscribe send e-mail to majordomo@T10.org with ‘subscribe’ in message the body

To unsubscribe send e-mail to majordomo@T10.org with ‘unsubscribe’ in the message body

Internet address for distribution via T10 reflector: T10@T10.org

Document Distribution:
Global Engineering Telephone: 303-792-2181 or
15 Inverness Way East 800-854-7179
Englewood, CO 80112-5704 Facsimile: 303-792-2192

Revision history:

Revision 0:

Initial draft

Revision 1:

Converted to ISO/IEC style.

Edits agreed in 1/2000 T10 meeting.

Revision 2:

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.

Revision 3:

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.

ANSI (r)
NCITS.*:199x**

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 by an operating system using Object Based Storage commands. Objects designate entities in which computer systems store data. The purpose of the OSD abstraction is to assign to the storage device the responsibility for managing where data is located on the device.

Contents

1 Scope 1
2 Normative References 3
2.1 Approved references 3
2.2 References under development 3
3 Definitions 3
3.1 Terms 3
3.1.1 application client: 3
3.1.2 Block Based Storage device (SBC). 3
3.1.3 Heterogeneous. 4
3.1.4 Object. 4
3.1.5 Object Based Storage Device (OSD). 4
Object Storage Architecture (OSA). 4
3.1.7 Object Group. 4
3.1.8 Session. 4
3.1.9 Storage device. 4
3.1.10 Storage Management. 4
3.1.11 Storage Area Network (SAN). 5
3.2 Symbols and abbreviations 5
3.3 Keywords 5
3.3.1 expected: 5
3.3.2 mandatory: 5
3.3.3 may: 5
3.3.4 may not: 6
3.3.5 obsolete: 6
3.3.6 optional: 6
3.3.7 reserved: 6
3.3.8 shall: 6
3.3.9 should: 6
3.3.10 vendor-specific: 6
3.4 Conventions 6
3.4.1 General 6
3.4.2 Hierarchy diagram conventions 7
4 SCSI OSD Model 7
4.1 Overall Architecture 7
4.2 Elements of the example configuration 8
4.3 Description of the OSD Architecture 11
4.3.1 SCSI Model 11
4.3.2 Storage devi ce organization. 11
4.4 OSD Sessions 12
4.4.1 Objects 13
4.4.1.1 Object Types 13

4.4.1.2 Object Organization 14
4.4.1.3 Well-Known Objects 14
4.4.2 Object Group Object List (GOL) in an Object Group Control Object 15
4.5 Overview of OSA Operation 15
4.5.1 Preparing a device for OSD operation 15
4.5.2 Startup – Discovery and Configuration 15
Verification .... 16~~15~~
4.5.3 Accessing data on the OSD Example 16
4.5.4 OSD Mandatory Actions template 17
4.5.5 OSD Optional Action 18
4.6 Reservations 19
5 Data fields 21
5.1 Command format 21
5.2 Variable Length CDB Definition 22
5.2.1 Fields used in Actions and Responses 22
5.2.1.1 Service Action Code 22
5.2.1.2 Attribute Mask 22
5.2.1.3 Length 22
5.2.1.4 Object ID 23
5.2.1.5 Option Byte 1 23
5.2.1.6 Option Byte 2. 23
5.2.1.7 Object Group ID 23
5.2.1.8 Object Group Remaining Capacity 24
5.2.1.9 Source Storage device. 24
5.2.1.10 Session ID 24
5.2.1.11 Starting byte address 24
5.3 Attributes 24
5.4 Data Storage Policies 24
5.4.1 Classes of Object Attributes 25
5.4.2 Other Attributes 27
5.4.2.1 OSD Control Object (DCO) 27
5.4.2.2 Over-subscription 27
5.4.2.3 Object Group Control Object (GCO) 27
5.4.3 Setting Session Parameter Values 28
5.4.3.1 General Structure 28
5.4.3.2 Attribute-Setting Commands 28
5.4.3.3 Attribute-Retrieving Commands 29
Commands 29
6.1 Format OSD (Mandatory) 31
6.2 CREATE (Mandatory) 31
6.3 LIST (Mandatory) 33
6.4 OPEN (Optional) 33
6.5 READ (Mandatory) 35
6.6 WRITE (Mandatory) 37

6.7 APPEND (Mandatory) 38
6.8 FLUSH Object (Mandatory)..39
6.9 CLOSE (Optional) 40
6.10 REMOVE (Mandatory)...41
6.11 CREATE OBJECT GROUP (Mandatory) 42
6.12 REMOVE OBJECT GROUP (Mandatory)...42
6.13 IMPORT Object (optional) 43
6.14 GET ATTRIBUTES (Mandatory)..45
6.15 SET ATTRIBUTES (Mandatory)...46

Figures

Figure 2 - Comparison of traditional and OSD storage models 8

Figure 3 - Example OSA Configuration 9

Figure 4 - Basic SCSI hierarchy (reference only)..11

Figure 5 - Basic OSD hierarchy 12

Table 7 OSD Mandatory commands with field lengths 18

Table 8 - OSD Optional Action with field lengths 18

Table 11 - Option byte 1..23

Table 12 - Option byte 2..23

Table 13 - Potential Object Attributes..26

Table 14 - Potential Storage Device Control Object Attributes 27

Table 15 - Object Group Control Object Attributes...28

Table 16 - Parameter Modifiers 28

Table 17 - Commands for object based storage devices 30

Table 18 - Format OSD...31

Table 19 - Format OSD Response..31

Table 20 - CREATE Object Action 32

Table 21- Response to CREATE Object Action 33

Table 22 - OPEN Object Action..34

Table 23 - Response to OPEN Object Action...35

Table 24 - Read Action....36

Table 25 - Response to Read Object Action....37

Table 26 - WRITE Object Action....37

Table 27 - Response to WRITE Object Action....38

Table 28 - APPEND....39

Table 29 - APPEND to Object Response....39

Table 30 - FLUSH Object Operation....40

Table 31 - FLUSH Object Response....40

Table 32 - CLOSE Object Action....41

Table 33 - Response to CLOSE Object Action....41

Table 34 - REMOVE Object Action....41

Table 35 - REMOVE Object Response....42

Table 36 - CREATE OBJECT GROUP Action....42

Table 37 - Response to CREATE OBJECT GROUP....42

Table 38 - REMOVE OBJECT GROUP Action....43

Table 39 - Response to REMOVE OBJECT GROUP Action....43

Table 40 - Import Object Action....44

Table 41 - Response to Import Object Action....45

Table 42 - GET ATTRIBUTES Action....46

Table 43 – Response to GET ATTRIBUTE Action....46

Table 44 - SET ATTRIBUTES Action....47

Table 45 - Response to Set Attribute Action....47

Figure 1 CREATE Action Option byte 2....48

Figure 2 OPEN Action Options....49

Figure 3 WRITE Action Option byte 2....50

Figure 4 WRITE Action Option byte 2....51

Figure 5 APPEND Action Option byte 2....51

Figure 6 REMOVE Object Action Option byte 2 52

Figure 7 APPEND Action Option byte 2 53

Figure 8 Option byte 1 support for aggregation 60

Table 51 - Example 2: Read Aggregated Objects 66

Foreword

(This foreword is not part of American National Standard NCITS XXX-200X.)

This SCSI command set is designed to provide efficient peer-to-peer operation of input/output logical units by an operating system using Object Based Storage commands. The SCSI command set assumes an underlying command-response protocol.

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.

This standard contains four informative annexes that are not considered part of this standard.

Requests for interpretation, suggestions for improvement or addenda, or defect reports are welcome. They should be sent to the National Committee for Information Technology Standards (NCITS, ITI, 1250 Eye Street, NW, Suite 200, Washington, DC 20005.

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 ANSI NCITS XXX -200X 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 ~~Requester~~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.

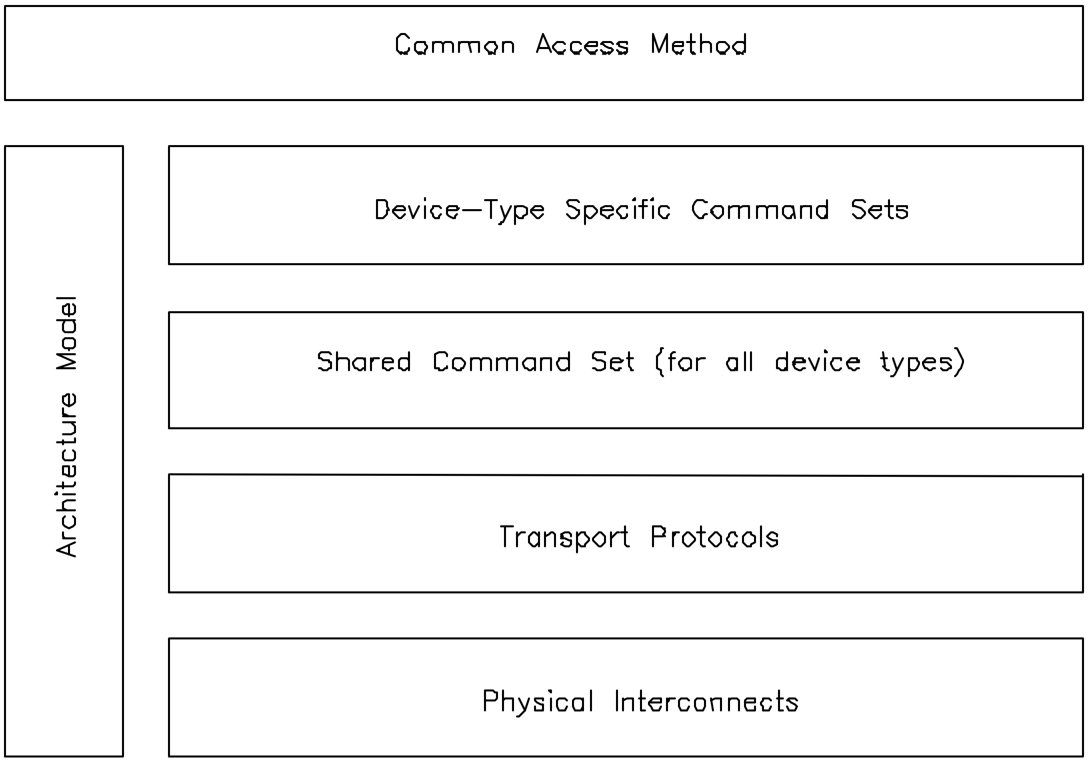

Figure 1 - SCSI standards - general structure

Figure 1~~Figure 1~~~~Figure 1~~ is intended to show the general structure of SCSI standards. The figure 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:

Physical Interconnects:

Fibre Channel Arbitrated Loop -2 [ANSI NCITS XXX –199X or 200X]

Fibre Channel - Physical and Signaling Interface [ANSI X3.230-1994]

High Performance Serial Bus [IEEE 1394-1995]

SCSI Parallel Interface -2 [ANSI NCITS 302-1999]

SCSI Parallel Interface -3 [ANSI NCITS XXX-200X]

Serial Storage Architecture Physical Layer 1 [ANSI X3.293-1996]

Serial Storage Architecture Physical Layer 2 [ANSI NCITS 307-1998]

Transport Protocols:

Serial Storage Architecture Transport Layer 1 [ANSI X3.295-1996]

SCSI-3 Fibre Channel Protocol [ANSI X3.269-1996]

SCSI-3 Fibre Channel Protocol - 2 [NCITS T10/1144D]

SCSI Serial Bus Protocol -2 [NCITS T10/1155D]

Serial Storage Architecture SCSI-2 Protocol [ANSI X3.294-1996]

Serial Storage Architecture SCSI-3 Protocol [ANSI NCITS 309-1998]

Serial Storage Architecture Transport Layer 2 [ANSI NCITS 308-1998]

Shared Command Set:

SCSI-3 Primary Commands [ANSI NCITS 301-1997]

SCSI Primary Commands – 2 [ANSI NCITS XXX-200X]

Device-Type Specific Command Sets:

SCSI Object Based Storage Device Commands (this standard)

SCSI-3 Block Commands [ANSI NCITS 306-1998]

SCSI-3 Enclosure Services [ANSI NCITS 305-1998]

SCSI-3 Stream Commands [NCITS T10/997D]

SCSI-3 Medium Changer Commands [NCITS T10/999D]

SCSI-3 Controller Commands [ANSI X3.276-1997]

SCSI Controller Commands - 2 [ANSI NCITS 318-1998]

SCSI-3 Multimedia Command Set [ANSI NCITS 304-1997]

SCSI Multimedia Command Set - 2 [NCITS T10/1228D]

Architecture Model:

SCSI-3 Architecture Model [ANSI X3.270-1996]

SCSI Architecture Model - 2 [NCITS T10/1157D]

Common Access Method:

SCSI Common Access Method [ANSI X3.232-1996]

SCSI Common Access Method - 3 [NCITS T10/990D]

The Small Computer System Interface -2 (ANSI X3.131-1994) is referred to herein as SCSI-2. The term SCSI in this standard refers to versions of SCSI defined since SCSI-2.

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.

2 Normative References

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

Copies of the following documents can be obtained from ANSI: Approved ANSI standards, approved and draft international and regional standards (ISO, IEC, CEN/CENELEC, ITU-T), and approved standards of other countries (including BSI, JIS, and DIN). For further information, contact ANSI's Customer Service Department at 212-642-4900 (telephone), 212-302-1286 (fax) or via the World Wide Web at http://www.ansi.org.

Additional availability contact information is provided below as needed.

2.1 Approved references

ANSI NCITS XXX.200X , Information technology - SCSI Primary Commands –2

ANSI NCITS XXX.200X , Information technology - SCSI Architecture Model –2

2.2 References under development

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

Editor's Note: This subclause may not be needed.

Note 1 - For more information on the current status of the document, contact the NCITS Secretariat at 202-737-8888 (telephone), 202-638-4922 (fax) or via Email at ncits@itic.org. To obtain copies of this document, contact Global Engineering at 15 Inverness Way East Englewood, CO 80112-5704 at 800-854-7179 (telephone), 303-792-2181 (telephone), or 303-792-2192 (fax).

3 Definitions

3.1 Terms

3.1.1 application client:

An object that is the source of SCSI commands. Further definition of an application client may be found in the SCSI Architecture Model -2 (SAM-2).

Note 2 – 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.

3.1.2 Block Based Storage device (~~BBSD~~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.

3.1.3 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.

3.1.4 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.

3.1.5 Object Based Storage Device (OSD).

A storage device in which data is organized and accessed as objects.

3.1.6 ~~Object Based Storage~~ Object Storage Architecture (~~OBS~~OSA).

This term is used to describe a storage architecture employing OSD.

3.1.7 Object Group.

A subset of the Objects on a single OSD. The subset may have a capacity quota associated with it.

~~3.1.8Requester.~~

~~A node in a cluster or network of systems with an application client that submits a request for action by a storage device. The term Requester is used as a general description for systems including both clients and servers, as either could be directly connected to OBSD and impose workloads on it. A client with an application client putting its workload on a server, which in turn has an application client that submits I/O requests to storage, is not a Requester. The client's server is, as the server is the element actually engaging in I/O with the OBSD.~~

~~3.1.9~~3.1.8 Session.

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

~~3.1.10~~3.1.9 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 Object Based Storage device may be any of these.

~~3.1.11~~3.1.10 Storage Management.

The task of enabling, controlling and maintaining physical storage (e.g., disk, tape, optical storage) 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.

~~3.1.12~~3.1.11 Storage Area Network (SAN).

A peer connection between one or more storage devices and one or more computers.

3.2 Symbols and abbreviations

CDB	command descriptor block
HSM	Hierarchical Storage Manager
I/O	input/output
ID	identifier
LSB	least significant bit
MMC	SCSI-3 Multimedia Command Set
MSB	most significant bit
OBS	Object Based Storage
OSA	Object Storage Architecture
OSD	Object Based Storage Device Commands
QoS	Quality of Service
SAM	SCSI-3 Architecture Model
SCC	SCSI-3 Controller Commands
SCSI	either SCSI-2 or SCSI-3
SCSI-2	the Small Computer System Interface-2
SCSI-3	the Small Computer System Interface-3
SPC	SCSI-3 Primary Command Set standard
XOR	exclusive-or

~~3.2~~3.3 Keywords

Keywords to differentiate levels of requirements and optionality.

~~3.2.1allowed:~~

~~Commands issued by initiators not holding the reservation or by initiators not registered when a registrants only persistent reservation is present should complete normally.~~

~~3.2.2conflict:~~

~~Commands issued by initiators not holding the reservation or by initiators not registered when a registrants only persistent reservation is present shall not be performed and the device server shall terminate the command with a RESERVATION CONFLICT status.~~

~~3.2.3~~3.3.1 expected:

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 that comply with the interoperability requirements of this standard

~~3.2.4~~3.3.2 mandatory:

Indicates items required to be implemented as defined by this standard.

~~3.2.5~~3.3.3 may:

Indicates flexibility of choice with no implied preference.

3.3.4 may not:

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

~~3.2.6~~3.3.5 obsolete:

Indicates items that were defined in prior SCSI standards but have been removed from this standard. ~~(Editor's note: Probably this one will be omitted since this is a first standard and therefore is not obsoleting anything in a prior standard.)~~

3.3.6 optional:

Describes features that are not required to be implemented by this standard. However, if any optional feature defined by the standard is implemented, it shall be implemented as defined by this standard.

~~3.2.8~~3.3.7 reserved:

Refers to bits, bytes, words, fields, and code values that are set aside for future standardization. Their use and interpretation may be specified by future extensions to this or other standards. A reserved bit, byte, word, or field shall be set to zero, or in accordance with a future extension to this standard. The recipient may not check reserved bits, bytes, words, or fields. Receipt of reserved code values in defined fields shall be treated as an error.

~~3.2.9~~3.3.8 shall:

Indicates a mandatory requirement. ~~Unless part of an optional feature that is not implemented, d~~Designers are required to implement all such mandatory requirements to ensure interoperability with other standard conformant products.

~~3.2.10~~3.3.9 should:

Indicates flexibility of choice with a strongly preferred alternative. Equivalent to the phrase "it is recommended."

~~3.2.11~~3.3.10 vendor-specific:

Items (e.g., a bit, field, code value, etc.) that are not defined by this standard and may be vendor defined.

~~3.3~~3.4 Conventions

3.4.1 General

Lower case is used for words having the normal English meaning. 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 clause 3or in the text where they first appear.

Listed items in this standard do not represent any priority. Any priority is explicitly indicated. Formal lists (e.g., (a) red; (b) blue; (c) green) connoted by letters are in an arbitrary order. Formal lists (e.g., (1) red; (2) blue; (3) green) connoted by numbers are in a required sequential order.

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

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).

The additional conventions are:

The names of abbreviations, commands, and acronyms used as signal names are in all uppercase (e.g., IDENTIFY DEVICE);

Fields containing only one bit are referred to as the "NAME" bit instead of the "NAME" field;

Field names are in SMALL CAPS to distinguish them from normal English;

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 immediately followed by lower-case h (xxh) are hexadecimal values;

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

If a field is specified as not meaningful or it is to be ignored, the entity that receives the field shall not check that field.

3.4.2 Hierarchy diagram conventions

Hierarchy diagrams show how entities are related to each other. In the corresponding hierarchy diagram, labeled boxes denote entities. The composition and relation of one entity to others is shown by the connecting lines. The I-beam symbol denotes a relationship where an entity contains either one or the other or both of a pair of other entities. In the hierarchy diagram, entities that are required to have one and only one instance are shown as simple boxes. The hierarchy diagram also shows multiple instances of an object by the presence of a shadow. Entities that are optional are indicated by light diagonal lines. An entity that may not have any instances, have only one instance, or have multiple instances is shown with both diagonal lines and a shadow. The instance indications shown in a hierarchy diagram are approximate, detailed requirements appear in the accompanying text. For a more detailed description of hierarchy diagrams see SAM-2.

4 SCSI OSD Model

4.1 Overall Architecture

The 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. See Figure 2~~Figure 2~~~~Figure 2~~.

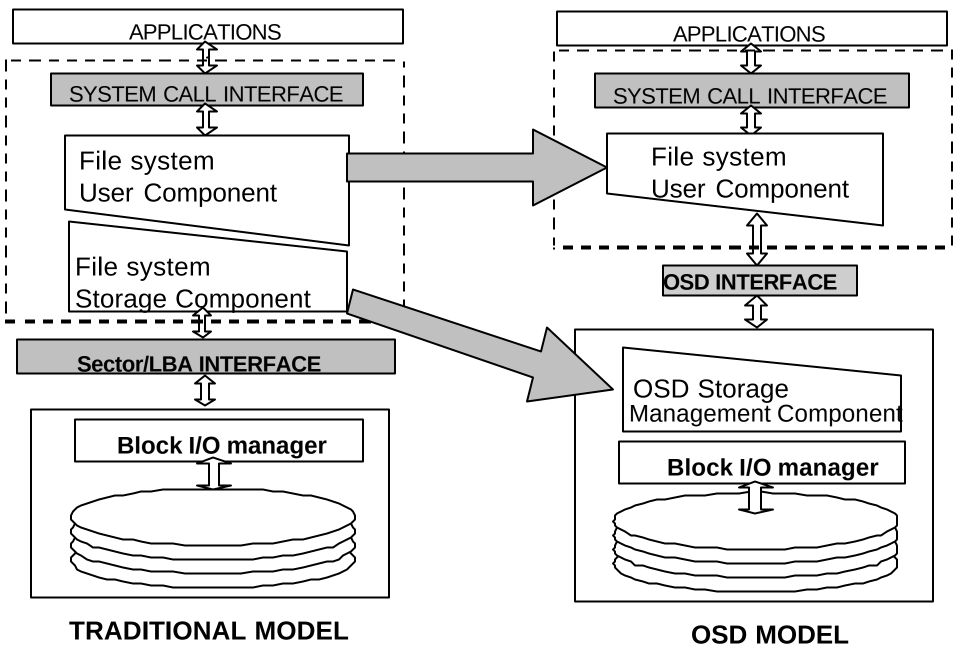

Figure 2 - Comparison of traditional and OSD storage models

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

Hierarchy management;

Naming;

User access control.

Although the storage management component is focused on mapping the file system logical constructs to the physical organization of the storage media, the file system will continue to have the ability to influence the properties of data through the specification of attributes. These 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 – such as on the outer zone of a disc drive to get higher data rate.

We have seen over the last several years many sub-components of storage management move to the storage device, such as 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 ~~Object Based Storage~~The Object Storage Architecture (~~OBS~~OSA) is to enable the sharing of storage in a heterogeneous processor cluster. This is more complex than simply defining a new protocol. The following illustrates how a protocol supporting the object abstraction might fit into an overall architecture. See Figure 3~~Figure 3~~~~Figure 4~~.

In this example, the ~~OBS~~OSA architecture has three, plus a potential fourth, constituents: OSD, Storage Area Network (SAN), and ~~Requesters~~host systems, a potential fourth element may be a dedicated Policy/Storage Manager.

The Object Based Storage Devices are the storage components of the system to be shared~~.~~ (e.g., ~~They include~~ disc drives, RAID subsystems, tape drives, tape libraries, optical drives, jukeboxes, or other storage devices~~) to be shared~~. They may have a SAN attachment with a path to the ~~Requesters~~ initiators that will access them.

The ~~Requesters~~ application clients are typically part of initiators within ~~are~~ the servers or clients sharing and directly accessing the OSD via a network. All I/O activity is between the ~~Requesters~~ application clients and the OSD. (A Policy/Storage Manager would also be an application client ~~Requester~~.)

The SAN or other interconnect is used by all ~~OBS~~OSA components to intercommunicate. It is assumed the SAN has the properties of both networks and channels.

A Policy/Storage Manager, if present, may perform management and security functions such as request authentication and aggregation management. The Policy/Storage Manager could relieve the ~~Requesters~~ other application clients of some storage management responsibilities. In other instances of ~~OBS~~OSA systems, a dedicated Policy/Storage Manager may not be needed. The equivalent function may be distributed among the ~~Requesters~~ application clients. This model ignores the role of the Policy/Storage Manager as it is not needed to describe how the commands work.

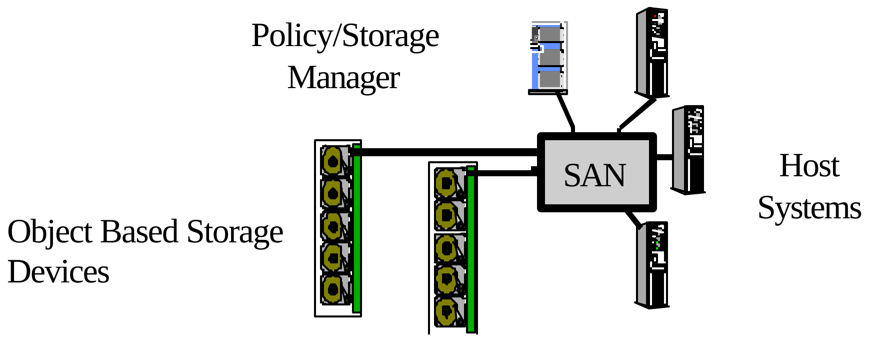

Figure 3~~3~~4 - Example ~~OBS~~ OSA Configuration

4.3~~Object Types~~

~~An OSD device is a logical unit within a SCSI device. As such, an OSD device will return the OSD Device Type value in response to an INQUIRY command. An OSD device contains only objects. Objects have two types of information associated with them:~~

~~a) Meta-data~~

~~b) Data~~

~~Meta-data describe specific characteristics or attributes of the object. This includes the size of the meta-data, the total amount of bytes occupied by the object (including meta-data), logical size of the object, as well as other parameters. The user data is contained in the object.~~

~~There are three different types of objects:~~

~~a) Root: this unique object is always present in the device. Its Meta-data contains device-global characteristics. 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 Group IDs. The root object is maintained by the OSD.~~

~~b) Group: this object is created by specific commands from an initiator. Its purpose is to contain a list of UserObjects that share some common attributes. Its meta-data contains its GroupID (a 32bit unsigned integer), the maximum number of UserObjects it may contain, the current number of UserObjects, the quota capacity of the group, the current capacity utilized by the group, as well as quality of service attributes common to all the objects in the group. The default attributes for a Group are inherited from the~~

~~attributes of the device (i.e., Root object). The Data component of a Group is the list of currently valid UserObject IDs. The Group object is maintained by the OSD.~~

c) ~~UserObject: these are the primary objects that contain user data. Consequently, the Data for this kind of object is user data; the OSD manages this data on behalf of the initiators. The Meta-data for a UserObject contains characteristics specific to the object. This includes the UserObject ID (a 64 bit unsigned integer), the logical size of the user data, and quality of service attributes. Default attributes for a UserObject are inherited from the attributes of the group in which it is contained.~~

~~When first shipped, there may only be a Root object. Its attributes are the "factory defaults". There may be no Groups or UserObjects such that the Data for the root object is empty. A FORMAT command to the OSD shall restore the device to this original state.~~

~~There is only one Root object per OSD. There may be many Group objects (up to the capacity of the OSD). The Root and Group objects may be called "well-known" in that the structure of the meta-data and data associated with these objects is predefined.~~

~~Additionally, for the Root object, the GroupID and ObjectID are predefined (zero in both cases). For the Group object, the GroupID is that of the group itself (it is assigned by the OSD when the group is created), and the ObjectID is predefined (zero). Only UserObjects contain arbitrary data (the content of this data is owned by the initiators). UserObjects have a GroupID of the group they belong to. Their ObjectID is that assigned by the OSD when the object is created.~~

~~Meta-data attributes for an object may be queried by the GET_ATTRIBUTE service action and may be changed by the SET_ATTRIBUTE service action.~~

~~To get a list of the valid GroupIDs, an initiator may do a LIST service action against the Root object. To get a list of the ObjectIDs in a group, the initiator may do a LIST service action against the Group object. READ and WRITE service actions to these objects are not allowed.~~

~~READ/WRITE/APPEND service actions are used to interface with the data of a UserObject.~~

~~4.4Sessions~~

~~A session is a set of state information maintained in the OSD for the purposes of setting parameters for data transfer. Sessions are used only for Read and Write (including Append) type actions of user data. Except for the default session, sessions are not persistent across a reset event or a power off cycle.~~

~~Every OSD has a default session, that determines the parameters of its underlying data transfer machinery. The SessionID of the default session is zero. Optionally, an OSD device may support other session parameters. The SessionID of any session other than the default session shall be non-zero. The SessionIDs are created by the OSD and provided to the initiator in returned data of an OPEN_SESSION service action. The parameters that govern a session may be queried (GET_SESSION_PARAMS) and changed SET_SESSION_PARAMS).~~

~~Read and Write service actions shall be handled only under an existing session. That is, such actions cannot create a session, they can only fall under the auspices of an existing one.~~

~~A CLOSE_SESSION is used to close one or all non-default sessions.~~

~~The number of sessions that an OSD device supports is vendor-specific (but it shall be at least one, for the default session).~~

~~Once a session is established, read/write actions may be requested within the context of that session.~~

~~One parameter of a session may be an expiration time. Consequently, some sessions may close automatically. The close of a session or the change of parameters for a session shall not affect the data transfer for any read/write action already being serviced by the OSD within that session. Only new read/write actions specifying that SessionID shall be affected.~~

~~4.5~~4.3 Description of the OSD Architecture

4.3.1 SCSI Model

Figure 4~~Figure 4~~ is shown for reference only. For normative description of the SCSI Model see SAM-2.

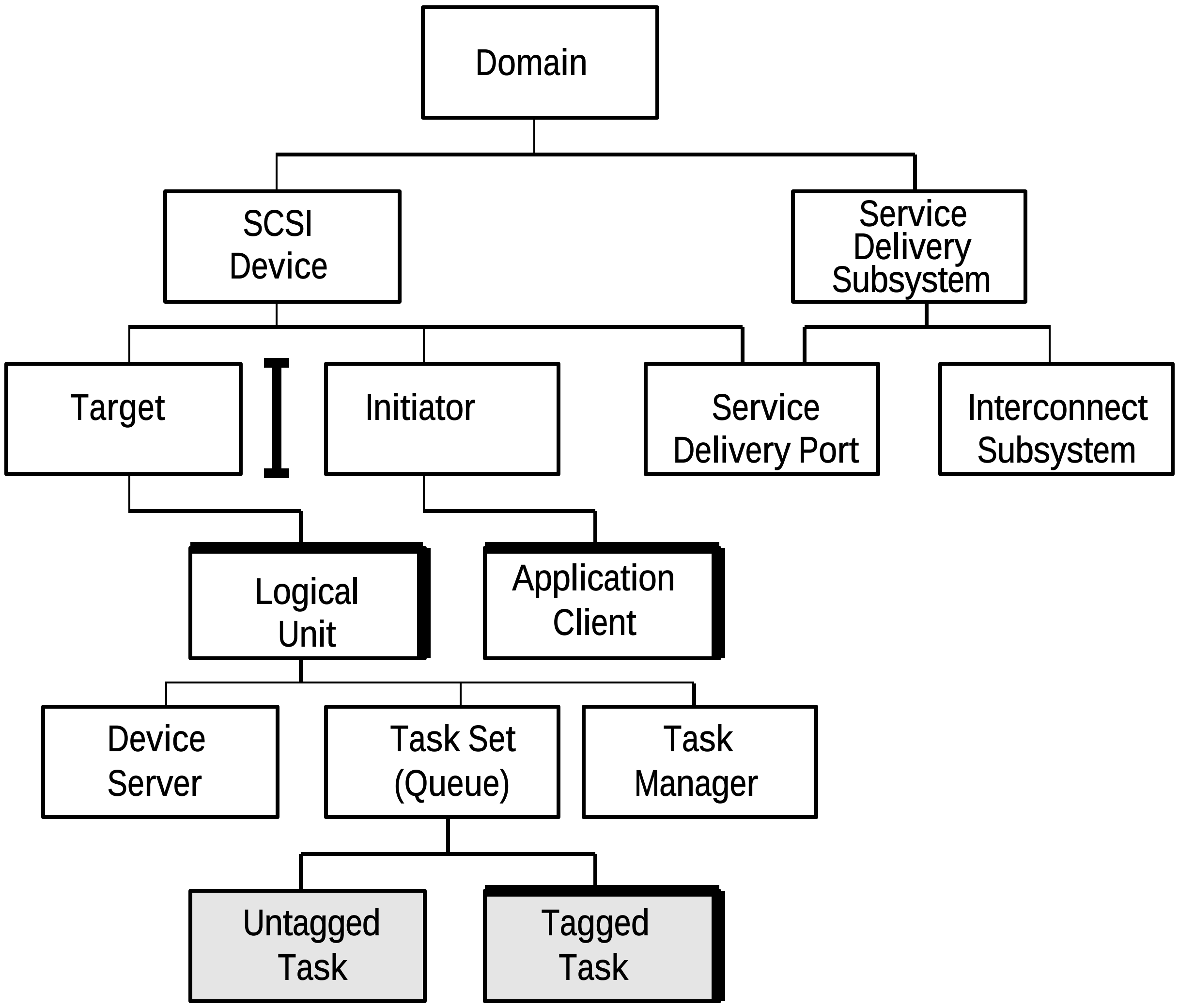

Figure 44 - Basic SCSI hierarchy (reference only)

~~4.5.1~~4.3.2 Storage device organization

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 ~~requester~~ application client a unique identifier that the ~~requester~~application client will use to access the data. The identifier is an unsigned integer that the storage device uses to connect 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. There is no notion of naming, hierarchical relationships, streams or file system style ownership and access control done within the object abstraction. The omitted features are assumed still to be the responsibility of the OS file system.

Editor's note: There are some access restrictions that are being included. Should there be an exception to some form of file level access control that is not included?

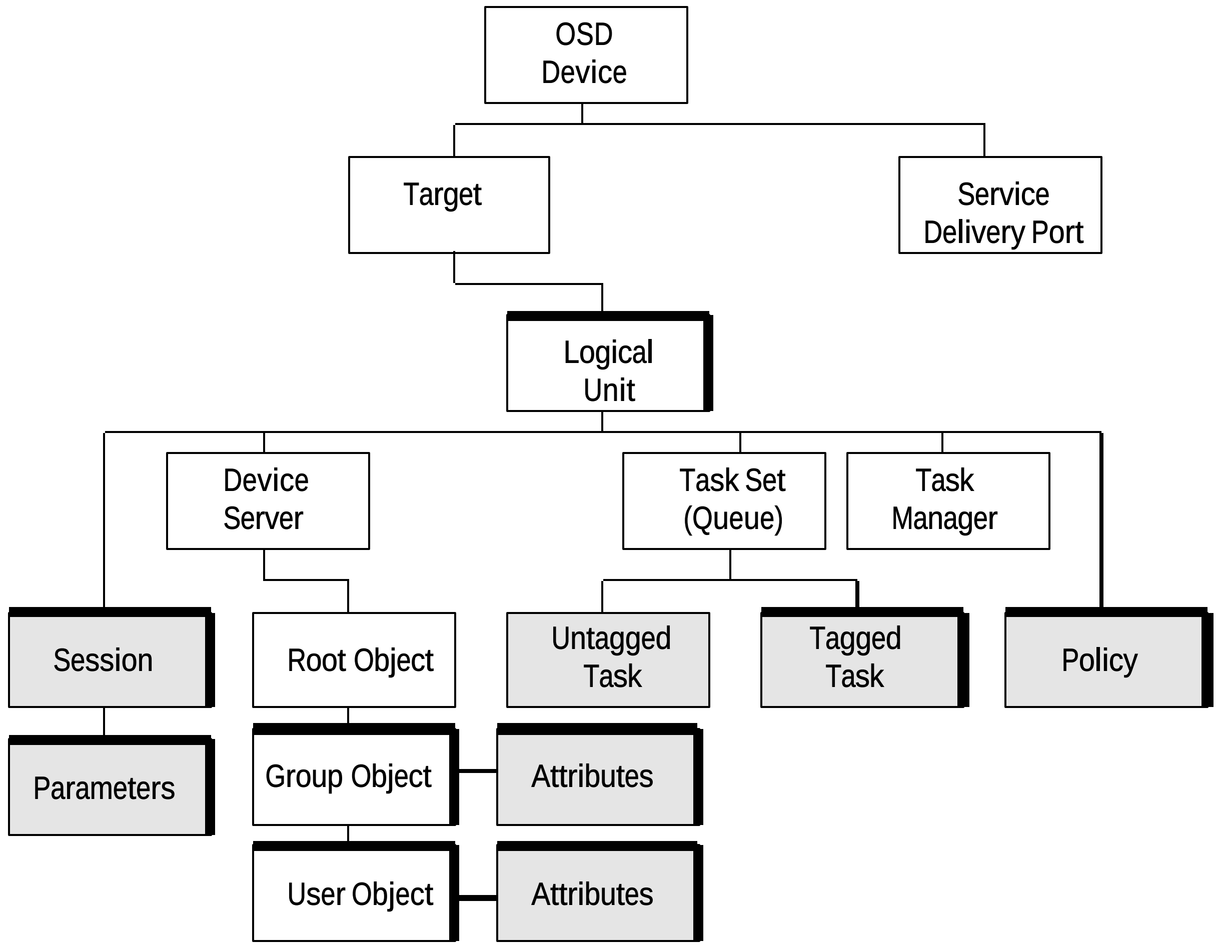

Figure 5~~5~~ - Basic OSD hierarchy

4.4 OSD Sessions

A session is a set of state information maintained in the OSD for the purposes of setting parameters for data transfer. Sessions are used only for Read and Write (including Append) type actions of user data. Except for the default session, sessions are not persistent across a reset event or a power off cycle.

Every OSD has a default session, that determines the parameters of its underlying data transfer machinery. The SessionID of the default session is zero. Optionally, an OSD device may support other session parameters. The SessionID of any session other than the default session shall be non-zero. The SessionIDs are created by the OSD and provided to the initiator in returned data of an OPEN_SESSION service action. The parameters that govern a session may be queried (GET_SESSION_PARAMS) and changed SET_SESSION_PARAMS).

Read and Write service actions shall be handled only under an existing session. That is, such actions cannot create a session, they can only fall under the auspices of an existing one.

A CLOSE_SESSION is used to close one or all non-default sessions.

The number of sessions that an OSD device supports is vendor-specific (but it shall be at least one, for the default session).

Once a session is established, read/write actions may be requested within the context of that session.

One parameter of a session may be an expiration time. Consequently, some sessions may close automatically. The close of a session or the change of parameters for a session shall not affect the data transfer for any read/write action already being serviced by the OSD within that session. Only new read/write actions specifying that SessionID shall be affected.

~~4.5.2~~4.4.1 Objects

4.4.1.1 Object Types

An OSD device is a logical unit within a SCSI device. An OSD device returns the OSD device type value in response to an INQUIRY command. An OSD device contains only objects not logical blocks. All objects contain attributes.

Attributes describe specific characteristics of the object. This includes the size of the attributes, the total amount of bytes occupied by the object (including attributes), logical size of the object, as well as other parameters. The user data is contained in the object.

There are three different types of objects:

a) Root: this unique object is always present in the device. Its attributes contains device-global characteristics. 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 Group IDs. The root object is maintained by the OSD.

b) Group: this object is created by specific commands from an initiator. Its purpose is to contain a list of UserObjects that share some common attributes. Its attributes contains its GroupID (a 32bit unsigned integer), the maximum number of UserObjects it may contain, the current number of UserObjects, the quota capacity of the group, the current capacity utilized by the group, as well as quality of service attributes common to all the objects in the group. The default attributes for a Group are inherited from the attributes of the device (i.e., Root object). The Data component of a Group is the list of currently valid UserObject IDs. The Group object is maintained by the OSD.

c) UserObject: these are the primary objects that contain user data. Consequently, the Data for this kind of object is user data; the OSD manages this data on behalf of the initiators. The Attributes for a UserObject contains characteristics specific to the object. This includes the UserObject ID (a 64 bit unsigned integer), the logical size of the user data, and quality of service attributes. Default attributes for a UserObject are inherited from the attributes of the group in which it is contained.

When first shipped, there may only be a Root object. Its attributes are the “factory defaults”. There may be no Groups or UserObjects such that the Data for the root object is empty. A FORMAT command to the OSD shall restore the device to this original state.

There is only one Root object per OSD. There may be many Group objects (up to the capacity of the OSD). The Root and Group objects may be called “well-known” in that the structure of the attributes and data associated with these objects is predefined.

Additionally, for the Root object, the GroupID and ObjectID are predefined (zero in both cases). For the Group object, the GroupID is that of the group itself (it is assigned by the OSD when the group is created), and the ObjectID is predefined (zero). Only UserObjects contain arbitrary data (the content of this data is owned by the initiators). UserObjects have a GroupID of the group they belong to. Their ObjectID is that assigned by the OSD when the object is created.

Attributes for an object may be queried by the GET_ATTRIBUTE service action and may be changed by the SET_ATTRIBUTE service action.

To get a list of the valid GroupIDs, an initiator may do a LIST service action against the Root object. To get a list of the ObjectIDs in a group, the initiator may do a LIST service action against the Group object. READ and WRITE service actions to these objects are not allowed.

READ/WRITE/APPEND service actions are used to interface with the data of a UserObject.

There are four attribute fields needed to establish access to data in an object. See Table 1~~Table 1~~~~Table 1~~

Table 1 - Addressing bytes in an object

Field	Size	Usage
OBJECT GROUP ID	4	Identifies in which subset of the Device's Objects' the desired object resides
OBJECT ID	8	Calls out the object being accessed
DISPLACEMENT	8	Identifies the starting location within the object of the data transfer
LENGTH	8	Number of bytes to be transferred

Addressing is in bytes. The objects on a storage device are grouped into sets. There may be one or more groups for the entire device.

~~4.5.3~~4.4.1.2 Object Organization

OSD Objects are collected into groups, called Object Groups. Optionally, there may be a capacity quota associated with a group, such that the OSD will ensure that the sum of storage capacity occupied by all the objects in a group does not exceed the capacity quota. The OSD device can be directed to reject WRITE, CREATE or APPEND operations to an object that would result in an object group consuming more storage capacity than it has been assigned. The capacity quota lets several independent ~~requesters~~ application clients, for instance, to be at work filling an OSD device with objects, without the danger of any consuming more than an allocated percentage of the available capacity.

~~4.5.4~~4.4.1.3 Well-Known Objects

A well-known Object is one that always has a specific object ID. A well-known Object shall exist on every storage device or in every Object Group. These objects serve as landmarks for an application client ~~or Requester~~ navigating the OSD organization. Table 2~~Table 2~~~~Table 3~~ includes example Object identifiers associated with each for illustration purposes.

Editor's note: Does the well-known object "exist in every storage device or in every Object Group" of does it "exist in every storage device and in every Object Group"

Table 22~~3~~ - Well Known Objects

Object Group ID	Object ID	Usage
0	1	~~Storage Device Control~~ Root Object
N	1	~~Object~~ Group ~~Control~~ Object

~~4.5.5~~4.4.2 Object Group Object List (GOL) in an Object Group Control Object

When an Object Group is created, a second well-known Object will also be built - the point of departure for navigating through the Objects. It will have the same identifier in every Object Group. The Object Group Control Object as part of its data includes a list of the Object ID's for all Objects resident in this Object Group.

Editor's Note: T10 should evaluate the usage of zero values for fields as we progress. The 9/14/2000 WG decided a zero ID for the Root Object is OK.

~~4.6~~4.5 Overview of ~~OBS~~OSA Operation

~~4.6.1~~4.5.1 Preparing a device for OSD operation

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 Table 3~~Table 3Table 5~~ to initialize and OSD device.

Table 33~~5~~ - Initialization Sequence

Operation	Parameters	Notes
Format OSD	LENGTH (optional)	Construct OSD control structures
CREATE OBJECT GROUP	Capacity Quota (optional)	Initialize Set into which Objects may be created

Upon completion of these two commands the storage device is an OSD device and shall accept other OSD mandatory commands and may accept OSD optional commands.

~~4.6.2~~4.5.2 Startup – Discovery and Configuration

Startup, discovery, and configuration techniques are a function of the interconnect protocols. When the ~~OBS~~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 ~~OBS~~OSA, the OSD devices, Policy/Storage Manager (if any), and ~~Requesters~~ other application clients would log onto the fabric. From the fabric they may learn of the existence of all other ~~OBS~~OSA components. They may use these fabric services to identify all other components. The ~~Requesters~~ 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 ~~OBS~~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 ~~Requester~~application client may need to be informed of the introduction of additional storage subsequent to startup, noted by an attribute set when the ~~Requester~~ 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.

~~4.6.2.1~~4.5.2.1 ~~Object Based Storage devices~~Verification

~~OSD devices have two important functions at start up. First, they log on to the network. Second, they shall not perform unauthorized activity. Both a requester and an OSD device may complete startup before an optional Policy/Storage Manager has had time to provide to the OSD device access control information. This leaves the OSD exposed to otherwise prohibited access. To prevent this, the OSD shall retain in non-volatile memory sufficient information to prevent unauthorized early access.~~

~~Editor's note: Are the Persistent Reservation requirements sufficient for the last requirement? If not where is the "sufficient information" defined? Does the "sufficient information" have to be accessible before the media is ready.~~

The OSD device shall reject invalid I/O requests With a CHECK CONDITION status and a sense key of ILLEGAL REQUEST. All ~~OBS~~OSA elements shall collaborate to enforce the security of the system. The details of the security collaboration are TBD or beyond the scope of this standard.

Editor's note: Are we going to take on the TBD or make the whole issue beyond the scope?

~~4.6.2.2Requesters~~

~~Requesters shall identify themselves to the network so that the optional Policy/Storage Manager may locate them and communicate to them sufficient direction to start storage access. The details of the identification process is beyond the scope of this standard. Depending on the security practice of a particular installation, A Requester may be denied access to some equipment. From the set of accessible storage devices it may then locate the files, databases, and free space available.~~

~~Editor' note: Should this last sentence be deleted?~~

~~4.6.3~~4.5.3 Accessing data on the OSD Example

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 4~~Table 4~~~~Table 7~~ 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.

Table 44~~7~~ - OSD command sequence for creating a file

Step	Operation	Group, Object	Notes – What file system does with the data
1	Get Attribute	n,1	Get Object ID of root object = object id "r"
2	READ	n,r	Make sure "father" does not already exist.
3	CREATE	n	Returns object "s", which will hold file "son"
4	CREATE	n	Returns object "f", which will 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"

The version of the CREATE action used in this example includes the actual transfer of data to the new object. See Table 5~~Table 5~~~~Table 9~~. Separate WRITEs are not need to fill "son" or "father" with data.

Table 5~~5~~~~9~~ - OSD command sequence using CREATE with data

Step	Operation	Group, Object	Notes – What file system does with the data
1	Get Attribute	n,1	Get OBJECT ID of ROOT OBJECT = OBJECT ID “r”
2	READ	n,r	Make sure “father” does not already exist.
3	CREATE	n	Returns object “s”, which holds file “son”
4	CREATE	n	Returns object “f”, which holds directory “father”
5	WRITE	n,r	Root directory revised to contain “father”

Error! Reference source not found. is an example that includes OPEN and CLOSE actions. These could be used to “lock” the root directory while it is being updated with the new directory “father”.

Table 6~~6~~~~11~~ - OSD command sequence with OPEN and CLOSE actions

Step	Operation	Group, Object	Notes – What file system does with the data
1	Get Attribute	n,1	Get OBJECT ID of root object = object id “r”
2	OPEN	n,r	Returns SESSION ID, QoS is “lock” this object ???
3	READ	n,r	Make sure “father” does not already exist.
4	CREATE	n	Returns object “s”, which will hold file “son”
5	CREATE	n	Returns object “f”, which will 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

~~4.6.4~~4.5.4 OSD Mandatory Actions template

Table 7~~Table 7~~~~Table 12~~ lists the action codes mandatory for OSD devices:

Table ~~77~~12 OSD Mandatory commands with field lengths

Field / Action	Options	Object Group	Object ID	Session ID	Starting byte	Byte Length	Attribute Mask	Reserved
	Bytes	Bytes	Bytes	Bytes	Bytes	Bytes	Bytes	Bytes
Format OSD	2					8		4
CREATE Object	2	4		4 (o)	8	8	8 (o)	
OPEN	2	4	8	4 (o)			8 (o)	
READ	2	4	8	4 (o)	8	8		
WRITE	2	4	8	4 (o)	8	8		
APPEND	2	4	8	4 (o)		8		
FLUSH Object	2	4	8					
CLOSE	2	4	8	4 (o)				
REMOVE	2	4	8					
CREATE OBJECT GROUP	2							
REMOVE Object Gr.	2							
Get Attributes	2	4	8	4 (o)			8	
Set Attributes	2	4	8	4 (o)			8	

Notes:

1) The (o) indicates fields that have optional meaning depending on a specified option.

The length field on the Format OSD action contains the amount of capacity to be allocated to the OSD.

OPEN and CLOSE as an object frame a session composed of a set of actions requiring additional management support or quality of service.

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 ~~requesters~~ 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.

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, object groups, and OSD themselves.

~~4.6.5~~4.5.5 OSD Optional Action

Editor's note: Should Table 8 OSD Optional Action include a Source Object ID?

The Import Object command enables an ~~Requester~~ application client to instruct an OSD device to read an Object from a second OSD creating and writing a copy onto itself.

Table ~~88~~14 - OSD Optional Action with field lengths

Field / Action	Options	Object Group	Object ID	Session ID	Starting byte	Byte Length	Source OSD	Source Group
IMPORT OSD	2	4	8	4	8	8	16	4

~~4.7~~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 session parameters or by the PERSISTENT RESERVE IN and ~~RELEASE~~ PERSISTENT RESERVE OUT commands (see ANSI NCITS TBD SPC-3). The OSD does not support the RESERVE and RELEASE commands.

These ~~RESERVE and RELEASE~~ 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.

~~Reservations are further controlled by the optional PERSISTENT RESERVE IN and PERSISTENT RESERVE OUT commands. For the requirements of this standard, reservations and releases made by use of the PERSISTENT RESERVE IN and PERSISTENT RESERVE OUT commands are the same as those using the RESERVE and RELEASE commands. See the ANSI NCITS TBD SPC-3 standard for a description and the requirements of the various reservation commands.~~

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 9~~Table 9~~~~Table 15~~. For the reservation restrictions placed on commands for the Reserve/Release management method see Table 9~~Table 9~~~~Table 15~~ column [A]. For the reservation restrictions placed on commands for the Persistent Reservations management method, see the columns under [B] in Table 9~~Table 9~~~~Table 15~~.

Commands from initiators holding a reservation should complete normally. The behavior of commands from registered initiators when a registrants only persistent reservation is present is specified in Table 9~~Table 9~~~~Table 15~~. A command that does not explicitly write the medium shall be checked for reservation conflicts before the command enters the current task state for the first time. Once the command has entered the current task state, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation. A command that explicitly writes the medium shall be checked for reservation conflicts before the device server modifies the medium or cache as a result of the command. Once the command has modified the medium, it shall not be terminated with a RESERVATION CONFLICT due to a subsequent reservation.

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

Table ~~99~~15— OSD commands that are allowed in the presence of various reservations

Command	~~Addressed LU is reserved by another initiator [A]~~	Addressed LU has this type of persistent reservation Held by another initiator ~~[B]~~				
		From any initiator		From registered initiator (RO all types)	From initiator not registered	
		Write Excl	Excl Access		Write Excl - RO	Exclusive Access - RO
APPEND		Conflict	Conflict	Allowed	Conflic t	Conflict
CLOSE		Allowed	Conflict	Allowed	Allowed	Conflict
CREATE		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		Conflict	Conflict	Allowed	Conflict	Conflict
INQUIRY		Allowed	Allowed	Allowed	Allowed	Allowed
LIST		Allowed	Conflict	Allowed	Allowed	Conflict
LOG SELECT		Conflict	Conflict	Allowed	Conflict	Conflict
LOG SENSE		Allowed	Allowed	Allowed	Allowed	Allowed
MODE SELECT(10)		Conflict	Conflict	Allowed	Conflict	Conflict
MODE SENSE(10)		Conflict	Conflict	Allowed	Conflict	Conflict
OPEN		Allowed	Conflict	Allow ed	Allowed	Conflict
PERSISTENT RESERVE IN		Allowed	Allowed	Allowed	Allowed	Allowed
PERSISTENT RESERVE OUT (REGISTER)		Allowed	Allowed	Allowed	Allowed	Allowed
PERSISTENT RESERVE OUT (CLEAR, PREMPT, RELEASE)		N/A	N/A	Allowed (1)	Conflict	Conflict
PERSISTENT RESERVE OUT (RESERVE)		Conflict	Conflict	Conflict	Conflict	Conflict
PREVENT-ALLOW MEDIUM REMOVAL (Prevent=0)		Allowed	Allowed	Allowed	Allowed	Allowed
PREVENT-ALLOW MEDIUM REMOVAL (Prevent<>0)		Conflict	Conflict	Allowed	Conflict	Conflict
READ		Allowed	Conflict	Allowed	Allowed	Conflict
READ BUFFER		Conflict	Conflict	Allowed	Conflict	Conflict
RECEIVE DIAGNOSTIC RESULTS		Conflict	Conflict	Allowed	Conflict	Conflict
REMOVE		Conflict	Conflict	Allowed	Conflict	Conflict
REMOVE OBJECT GROUP		Conflict	Conflict	Allowed	Conflict	Conflict
REPORT LUNS		Allowed	Allowed	Allowed	Allowed	Allowed
REQUEST SENSE		Allowed	Allowed	Allowed	Allowed	Allowed
SEND DIAGNOSTIC		Conflict	Conflict	Allowed	Conflict	Conflict
SET ATTRIBUTES		Conflict	Conflict	Allowed	Conflict	Conflict

(continued)

(concluded)

Command		Addressed LU has this type of persistent reservation Held by another initiator				
		From any initiator		From registered initiator (RO all types)	From initiator not registered	
		Write Excl	Excl Access		Write Excl - RO	Exclusive Access - RO
START/STOP UNIT START=1 and POWER CONDITION=0		Allowed	Allowed	Allowed	Allowed	Allowed
START/STOP UNIT START=0 or POWER CONDITION<>0		Conflict	Conflict	Allowed	Conflict	Conflict
OSD SYNCHRONIZE CACHE		Conflict	Conflict	Allowed	Conflict	Conflict
TEST UNIT READY		Conflict	Conflict	Allowed	Conflict	Conflict
WRITE		Conflict	Conflict	Allowed	Conflict	Conflict
WRITE BUFFER		Conflict	Conflict	Allowed	Conflict	Conflict

Key: ~~[A] = Reserve/Release, [B] = Persistent Reservations~~ LU = Logical Unit, Excl = Exclusive, RO = Registrants Only, <> Not Equal

Notes:

(1) Reservation is not released.

Allowed = Commands issued by initiators not holding the reservation or by initiators not registered when a registrants only persistent reservation is present should complete normally.

Conflict: Commands issued by initiators not holding the reservation or by initiators not registered when a registrants only persistent reservation is present shall not be performed and the device server shall terminate the command with a RESERVATION CONFLICT status.

~~Editor's note: This table will be filled in with all of the OSD commands.~~

Editor's note: The SPC commands should be deleted from this table before forwarding to prevent conflict and are shown for temporary reference only.

Note 3: When a system is integrated with more than one application client, agreement is required between the application clients as to how media is reserved and released during operations, otherwise, an application client may be locked out of access to a logical unit in the middle of an operation.

5 Data fields

5.1 Command format

The standard method for issuing SCSI I/O commands to storage devices involves a set of data grouped together in a Command Descriptor Block (CDB). 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.

A command is communicated by sending a command descriptor block to the device server. The OSD commands use the variable length CDB format (see SPC-3). The command descriptor block shall have an operation code as its first byte and a control byte as its second byte. The general structure of the operation code and control byte are defined in SAM-2. 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.

5.2 Variable Length CDB Definition

Table 10~~10~~~~17~~ - Variable Length CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (7Fh)							
1	Control							
2	Reserved							
3	Reserved							
4	Reserved							
5	ENCRYPTION IDENTIFICATION							
6	Reserved							
7	Additional CDB Length (n-7)							
8	(MSB)			Service Action				
9								(LSB)
10								
-				Service Action specific fields				
n								

The encryption identification field indicates whether CDB bytes 8 through n are encrypted. The value also indicates the encryption key to use for decryption. A value of zero indicates no encryption. The other values are TBD.

The additional CDB length field indicates the number of additional CDB bytes. This number shall be a multiple of 4.

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.

5.2.1 Fields used in Actions and Responses

The following fields are used in the OSD CDBs.

5.2.1.1 Service Action Code

This is a two byte field that uniquely identifies the operation to be performed.

5.2.1.2 Attribute Mask

This 64-bit field is a bit mask, with one bit for each page of attributes to be read or written. A bit set to one indicates to the OSD device that the corresponding attribute shall be written or supplied by the OSD device.

Editor's note: Needs further definition.

~~Editor's note: Should CRC for OSD service actions be defined and mandatory? A pseudo requirement was in a footnote.~~

5.2.1.3 Length

The length field is:

(1) an unsigned 64 bit integer representing the length of the data transfer supplied in the action by the ~~requester~~application client;

(2) the actual length of the transfer supplied by the OSD device in the response to the action or;

(3) the size in bytes of the storage device to be formatted as an OSD device.

5.2.1.4 Object ID

The Object ID is an unsigned 64 bit integer assigned by the OSD device.

5.2.1.5 Option Byte 1

Option Byte 1 is a set of fields used to modify or control the Action.

Table 11~~1118~~ - Option byte 1

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

5.2.1.6 Option Byte 2.

Option Byte 2 is a set of fields used to modify or control the Action. O2-0 through O2-7 specify conditions or qualifiers for each action as specified in this standard.

Table 12~~1219~~ - Option byte 2

Bit	7	6	5	4	3	2	1	0
Byte 0	O2-7	O2-6	O2-5	O2-4	O2-3	O2-2	O2-1	O2-0

5.2.1.7 Object Group ID.

This is an unsigned 32-bit integer assigned by the OSD device. There is a requirement with some actions (GET ATTRIBUTES, SET ATTRIBUTES, FLUSH and REMOVE OBJECT GROUP) to reference the entire OSD device, not just a single Object Group. The value 0 in this field references the entire OSD device.

5.2.1.8 Object Group Remaining Capacity.

This is an unsigned 64-bit integer supplied by the OSD device in response to WRITE, APPEND, IMPORT or CREATE actions and indicating the amount of capacity remaining in the space quota for the referenced Object Group.

5.2.1.9 Source Storage device.

This 16 byte field contains an OSD name (used externally to address the named OSD device). It is used in the IMPORT action to identify the source for an object whose data is to be imported into the requesting OSD device. Interconnect addressing conventions govern the translation of an OSD name into an interconnect-specific address.

5.2.1.10 Session ID.

This four-byte field binds a request to a previously granted quality of service agreement.

5.2.1.11 Starting byte address.

This is an unsigned 64-bit integer supplied by the ~~requester~~application client and indicating the location where the read or write is to commence in the specified object relative to the first byte (byte 0).

5.3 Attributes

Attributes are characteristics associated with objects to prescribe desired behaviors and/or effects upon object access, or to assign intrinsic properties to objects for use as ~~meta-data~~attributes. Since OSD objects are intended to contain data, an object's attributes may apply to its data as well. ~~The use of attributes may enable the exploitation of the OSD architecture by applications running at a remote host or within the OSD device itself.~~

~~Editor's note: Where is the concept of running applications in the OSD device underpinned? @@ This is the end of general editing in revision 2.~~

In some cases attributes specify OSD performance expectations. Benchmarks ~~will~~ may be used to enable OSD manufacturers to specify attribute ranges that are meaningful for their devices. ~~An auditing mechanism will need to exist to measure the device's actual performance for compliance purposes.~~

5.4 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 ~~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, ~~and~~ 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 ~~document~~standard.

Editor's note: Should we leave policies beyond the scope, add ~~withing~~within the scope a method to convey policies, or delete them altogether?

~~5.3.2~~5.4.1 Classes of Object Attributes

Four classes of attributes are ~~manifest~~specified: OSD-determined, static, session, and extended. The device sets OSD-determined attributes as a result of an operation on the object. Classes are particularly useful to policy-driven space maintenance processes, and include:

Nearby Object – Object ID of an Object as close as possible to which this object should be located;

Depending(??) Object – Object ID of an Object on which this object is dependent;

Copied Object – Object ID of Object ~~of which~~ that this object is a copy of;

Object logical length – The highest byte address that will return data on a READ request.

Editor's note: These classes need discussion and are there others not specified?.

Static attributes are persistent characteristics that are set by the process creating or updating the object. These attributes are ideally suited for use by host file systems, and include such object ~~meta-data~~attributes as:

—OBJECT_SIZE - The amount of storage space the OSD device associates with this object;~~.~~

—CREATED_TIME – the time the object was initially created;~~, based on the OSD device's clock~~

—DATA_MODIFIED_TIME – the time the object was last written into;~~, or edited, based on the OSD device's clock~~

—DATA_ACCESS_TIME – the time the object was last read or written into;~~, based on the OSD device's clock~~

—ATTRIBUTE_MODIFIED_TIME – the time the attributes of the object were last modified;~~.~~

—EXPIRATION _TIME_STAMP – The time when the object is no longer required;

—INDELIBLE – a 'non-modifiable' indicator;~~.~~

—TRANSIENT_OBJECT – This identifies an object that the file system does not expect to survive power ~~failures.~~off/on cycles.

—FILE SYSTEM – Identification of the file system under which the object was created;~~.~~

—FILE SYSTEM SPECIFIC ATTRIBUTES - File system-specific information, uninterpreted by the OSD device.

Editor's note: Who is going to maintain the File System registration? Should we delete the FILE SYSTEM attribute or just make it a host assignable code with no registration implication for T10?

Table 13~~13~~~~20~~ - ~~Possible~~ Potential Object Attributes

Type	Name	Set by	Length	Semantics
Clustering	Nearby_Object	Set Attribute	16	Locate this Object near another
Depending Object	Depending_Object	Set Attribute	16	This Object is dependent on the named object
Cloning	Copied_Object	OSD	8	Object was created Copy Object
	Object_Logical_Length	OSD, Set Attr.	8	Largest offset written
Size	Object_Size	OSD, Set Attr.	8	Number of Bytes Allocated for Object
Access control	Access control state	Set Attribute	2	Access version
			2	Reserved
	Created_Time	OSD, CREATE	8	Timestamp of object creation
	Data_Modified_Time	OSD, CLOSE	8	Timestamp of last object data modification
Time	Data_Accessed_Time	OSD, OPEN	8	Timestamp of last data access
	Attribute_Modified_-Time	OSD, Set Attr.	8	Timestamp of last attribute modification
	Expiration_Time_-stamp	CREATE, Set Attribute	8	Timestamp after which object is not required
Miscellaneous	Object_Attributes	OSD, Set Attribute	8	Bits of Object properties for self-mgmt 00: indelibility 01: transcient_object
File System	File_System_ID	Set Attribute	2	Identification of the OS creating the object
	FS-Specific	Set Attribute	256	U~~s u~~ninterpreted by OSD ~~??”s”??~~

Session attributes specify changeable characteristics that may be directly modified by the accessing process. The parameters may be set when the process establishes a session with the OSD device. **The value set** of the process-object dynamic parameters is maintained independently of other process-object sets, even if the same object is used; hence, the OSD device needs to relate a unique session-id to the **value set** associated with each process-object pair. Dynamic parameters include:

Editor’s note what does the value set statement refer to?

- —TIME TO INITIAL ACCESS (TIA) – The maximum time delay (in milliseconds) that can be tolerated until the first byte of data from the object is delivered.
- —SUSTAINED ACCESS RATE (SAR) – The on-going average data rate (in bytes per second) that data should be read from or written to the object.
- —FREQUENCY OF ACCESS (IOR) – The average number of requests per second that may be expected to read data from or write data into the object.
- —ACCESS REQUEST SIZE (ARS) – The average request size of data to be read from or written into the object.

Each of these dynamic parameters are specified separately for:

- —READ VS. WRITE PARAMETER – indicates the parameter is for read requests or for write requests
- —RANDOM VS. SEQUENTIAL ACCESS – indicates the specification is for random access requests or for sequential requests.

Extended parameters are TBD.~~not defined at this time, but the capability is provided to enable new and/or higher level parameters to be provided in the future. Provision is made in the command structure for supporting these parameters in the future (cf. Section 5.3.4.1).~~

~~5.3.3~~5.4.2 Other Attributes

It is recognized that both object groups and the OSD device ~~themselves could~~ may have attributes associated with them. They ~~could~~ may have default values for the contained objects as well as special properties that pertain to the larger entities. Access to these attributes is achieved by associating ~~special,~~ well known object ID's with object groups and the OSD device. In these cases the interpretation of an attribute mask and its values changes as below.

~~5.3.3.1~~5.4.2.1 OSD Control Object (DCO)

This object contains the attributes the OSD device shall maintain that relate to the storage device ~~itself~~ or that relate to all objects on the storage device. The attributes are maintained by the SET ATTRIBUTE function. Each logical unit has one DCO. Control objects are intended to serve, for ~~OBS~~OSA systems, a function similar to SCSI MODE SENSE and MODE SELECT.

Table 14~~1421~~ - Potential Storage Device Control Object Attributes

Attribute	Length	Semantics
NAME	8	Immutable identifier
USER NAME	8	Installation supplied name
CLOCK	8	Monotonic counter
MASTER KEY	16	master key, controlling storage device key
STORAGE DEVICE KEY	16	storage device key, controlling Object Group keys
PROTECTION LEVEL	n	defines protection options
OBJECT_GROUP_COUNT	4	Number of Object Groups on storage device
ATTRIBUTES	4	Properties of this Storage device 00 00 00 01: Over-subscription of capacity
OBJECT_ATTRIBUTES	n	Properties common to all objects on storage device

~~5.3.3.2~~5.4.2.2 Over-subscription

This attribute, if set, allows the Object Group quota to exceed the capacity of the device.

Editor's note: Yes. And then what? Is capacity prior to compression or post compression?

~~5.3.3.3~~5.4.2.3 Object Group Control Object (GCO)

This object contains the properties of a single Object Group. It describes not only the Object Group but also any object attributes that pertain to all objects in the Object Group. The OSD device ~~will~~ shall have one GCO for each Object Group defined on the storage device. Optional quotas ~~can~~ may be set to put limits on space associated with each ~~ID~~ object or the space used by all objects of the Object Group. Group Control ~~o~~Objects are intended to serve for ~~OBS~~OSA systems a function similar to SCSI MODE SENSE and MODE SELECT.

Table 15~~1522~~ - Object Group Control Object Attributes

Name	Length	Semantics
Group_Key	16	Encryption keys
Current_Working_Key	16	
Previous_Working_Key	16	
Object_Group_capacity_quota	8	Limit on sum of sizes of objects in this Object Group
Remaining_Capacity	8	Available Capacity in Group remaining against quota
Root_Object	8	Object ID of starting point for navigating objects
Object_size_limit	8	No object can extend beyond this length
Object_Attributes	n	Defines properties associated with all objects in Object Group

~~5.3.4~~5.4.3 Setting Session Parameter Values

~~5.3.4.1~~5.4.3.1 General Structure

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

Table 16~~1623~~ - Parameter Modifiers

Parameter Identifier	Comparator	Low/Only Value	High Value

Where:

- Parameter Identifier - indicates the particular parameter being set.~~;~~ ~~t~~The identifier enables discriminating among static, dynamic, and extended parameters;
- 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);~~.~~
- Low/Only Value – an integer representing the bottom of a range, if one is indicated by the comparator; otherwise, it is the unique value;
- High Value – an integer representing the top of a range, if one is indicated by the comparator; otherwise it is absent~~, otherwise~~.

Editor's note: Should absent be changed to zero?

Attribute retrieval command~~(~~s~~)~~ only ~~specify~~ utilize the attribute identifier.

~~5.3.4.2~~5.4.3.2 Attribute-Setting Commands

The following commands may be used to set object attributes:

—CREATE – sets static and dynamic attributes~~;~~. ~~d~~Dynamic attributes, if specified, become the defaults for any session or command that subsequently accesses this object;

—OPEN – sets dynamic attributes; if attributes are specified, this command creates a new session-identifier for the object;

—IMPORT OBJECT – sets static and dynamic attributes.~~;~~ ~~d~~Dynamic attributes, if specified, become the defaults for any session that subsequently accesses this object;

—SET OBJECT ATTRIBUTES – sets static and dynamic attributes.~~;~~ ~~i~~If a non-null session-identifier is specified, sets dynamic attributes for that session.

Return codes are used to communicate the success/failure of setting the appropriate attributes.

~~5.3.4.3~~5.4.3.3 Attribute-Retrieving Commands

The following commands may be used to retrieve object attribute settings:

—GET OBJECT ATTRIBUTES – all attributes may be ~~specified~~ requested.

TBD

The attributes and corresponding values are returned as a result of the commands.

6 ~~Actions (~~Commands~~)~~

The ~~following~~commands in Table 17~~Table 17~~ are the operations that an OSD device ~~will have to~~ may perform as its contribution to the ~~OBS~~OSA environment. The action field in the CDB uniquely identifies the operation to take place. ~~Each section~~ The command subclauses describe~~s~~ 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 ~~Requester~~application client is also listed to develop a clearer idea of what the function entails.

Only the unencrypted versions of the commands are described. ~~It is felt that more work must be done to define t~~The security architecture ~~before~~ specifying any ~~the~~ encryption and authentication fields ~~can be defined~~ is TBD.

In the CDB descriptions some fields are described as optional. The fields are always present. An optional field is one that may have a value of zero to indicate that it is not ~~to be used~~ implemented.

Editor's note: The zero statement needs to be verified for each command.

Table 17~~17~~ - Commands for object based storage devices

Command name	Operation code	Action Code	Type	Subclause
APPEND	7Fh	8807h	M	
CLOSE	7Fh	8809h	O	
CREATE	7Fh	8802h	M	
CREATE OBJECT GROUP	7Fh	880Bh	M	
Format OSD	7Fh	8801h	M	
FLUSH Object	7Fh	8808h	M	
GET ATTRIBUTES	7Fh	880Eh	M	
IMPORT	7Fh	880Dh	O	
INQUIRY	12h	N/A	M	SPC-3
LIST	7Fh	8803h	M	
LOG SELECT	4Ch	N/A	O	SPC-3
LOG SENSE	4Dh	N/A	O	SPC-3
MODE SELECT(10)	55h	N/A	O	SPC-3
MODE SENSE(10)	5Ah	N/A	O	
OPEN	7Fh	8804h	O	
PERSISTENT RESERVE IN	5Eh	N/A	M	SPC-3
PERSISTENT RESERVE OUT	5Fh	N/A	M	SPC-3
PREVENT-ALLOW MEDIUM REMOVAL	1Eh	N/A	O	SPC-3
READ	7Fh	8805h	M	
READ BUFFER	3Ch	N/A	O	SPC-3
RECEIVE DIAGNOSTIC RESULTS	1Ch	N/A	O	SPC-3
REMOVE	7Fh	880Ah	M	
REMOVE OBJECT GROUP	7Fh	880Bh	M	
REPORT LUNS	A0h	N/A	O	SPC-3
REQUEST SENSE	03h	N/A	M	SPC-3
SEND DIAGNOSTIC	1Dh	N/A	M	SPC-3
SET ATTRIBUTES	7Fh	880F	M	
START STOP UNIT	1Bh	N/A	O	SBC-2
OSD SYNCHRONIZE CACHE	TBD	TBD	O	TBD
TEST UNIT READY	00h	N/A	M	SPC-3
WRITE	7Fh	8806h		
WRITE BUFFER	3Bh	N/A	O	SPC-3

Key: M = Command implementation is mandatory.

O = Command implementation is optional.

OB = Obsolete

SPC = SCSI Primary Commands

Notes: All remaining OSD operation codes and action codes (8800h, 8810h-88ffh) are reserved for future standardization.

6.1 Format OSD (Mandatory)

This action code causes the ~~Storage D~~OSD device to delete all UserObjects, 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 ~~meta-data~~attribute structures being constructed that support the creation and access of objects.

Table 18~~18~~24 - Format OSD

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			FORMAT OSD ACTION CODE (8801h)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
24	(MSB)							
25				Reserved				
26								
27								(LSB)
28	(MSB)							
29								
30				LENGTH				
31				(OSD _CAPACITY)				
32								
23								
34								
35								(LSB)

If the OSD Length is set to 0, the entire device is formatted as an OSD device. Any other value specifies the total OSD device capacity in Bytes. If the value is greater than the maximum OSD device capacity the value is rounded down to the OSD device capacity for the format operation. OSD device capacity is the sum of the root object, group objects, and all potential user objects including attributes and user data after compression.

Editor's note: The definition of OSD device capacity has not yet been agreed to.

Table 19~~19~~~~25~~ - Format OSD Response

Bit / Byte	7	6	5	4	3	2	1	0
0				SCSI STATUS BYTE				

6.2 CREATE (Mandatory)

The CREATE causes the OSD device to allocate an unused OBJECT ID. The ~~Requester~~ application client uses this ~~when~~ once before issuing WRITEs to the new object. In addition, the ~~Requester~~ application client ~~can~~ may specify several optional~~s~~ attributes ~~it wants~~ for the object.

An optional ~~special~~ instance of this command includes all data associated with an Object, so that in one command an object ~~can~~ may be created, written and closed. The Length field is always present, though its content may be meaningful only when certain options are invoked.

Editor's note: Always present appears to be in conflict with the table.

Table 20~~2026~~ - CREATE Object Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			CREATE ACTION CODE (8802h)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13				OBJECT GROUP ID				
14								
15								(LSB)
16	(MSB)							
17								
18								
19				STARTING BYTE				
20				ADDRESS				
21								
22								
23								(LSB)
24	(MSB)							
25								
26								
27				LENGTH				
28				(optional)				
29								
30								
31								(LSB)
32	(MSB)							
33								
34				ATTRIBUTE				
35				MASK				
36				(optional)				
37								
38								
39								(LSB)

The information transfer includes the attributes, followed by object data, if present.

The response to the CREATE action includes the OBJECT ID allocated for the new Object. This is returned by the OSD device upon completion of the CREATE Action.

Table 21~~21~~~~27~~- Response to CREATE Object Action

Bit / Byte	7	6	5	4	3	2	1	0
0	SCSI STATUS BYTE							
1	(MSB)							
2								
3								
4								
5				OBJECT ID				
6								
7								
8								(LSB)
9	(MSB)							
10								
11								
12				OBJECT GROUP REMAINING CAPACITY				
13								
14								
15								
16								(LSB)
17	(MSB)							
18				SESSION ID				
19				(optional)				
20								(LSB)

6.3 LIST (Mandatory)

The LIST service action is used to get data from the Root or a Group object. This service action is a DataIn type service action and has the following syntax:

LIST(GroupID, Number, Index, [SortOrder], AllocationLength)

A zero GroupID refers to the Root object and non-zero GroupID refers to a valid Group object. Number specifies the number of IDs to be returned (GroupIDs if referencing the Root object and UserObject IDs otherwise). Index specifies the starting position of the IDs within the specified SortOrder, with initial position value of zero. SortOrder is optional. The default order is vendor specific. Support for specific SortOrder rules is TBD. The AllocationLength is the amount of space (in bytes) set aside in the DataIn buffer of the initiator. The returned data shall contain a header that specifies the sort order, the GroupID, and other additional data to allow for the returned parameter data to be self-parsable. (Details are TBD.)

The LIST service action shall ignore session parameters.

~~Editor's note: It does not seem appropriate to both fetch UserObjects and ignore session parameters.~~

6.4 OPEN (Optional)

The~~is~~ OPEN communicates to the OSD device that a certain object is to be accessed. It also indicates the kind of operations permitted on the object. The OPEN allows the ~~Requester~~ application client to prefetch read ~~and write~~ data for the specified Object. The ~~Requester~~ application client may optionally start an I/O session via the OPEN. All parameters are supplied. The LENGTH and SESSION ID fields may be set to zero, indicating they are not to be used. If non-zero, LENGTH indicates the number of bytes to be pre-allocated for this object. This also allows the ~~Requester~~ application client to caches writes, with confirmation that there will be available storage capacity to store the data when it is eventually sent to the OSD device. The OSD device pre-allocates capacity of this amount.

The SESSION ID establishes that I/O to this object are to have specific quality of service properties, those associated with this SESSION ID, ~~which~~ that had been returned to the ~~Requester~~ application client in a response to a previous OPEN Object. This lets an

~~Requester~~ application client ~~–~~associate the IO activity on several objects with a common session.

Table ~~222~~~~22~~~~28~~ - OPEN Object Action

Byte \ Bit	7	6	5	4	3	2	1	0
8	(MSB)	OPEN ACTION CODE (8804h)						
9								(LSB)
10	OPTION BYTE 1							
11	OPTION BYTE 2							
12	(MSB)	OBJECT GROUP ID						
13								
14								
15								(LSB)
16	(MSB)	OBJECT ID						
17								
18								
19								
20								
21								
22								
23								(LSB)
24	(MSB)	LENGTH						
25								
26								
27								
28								
29								
30								
31								(LSB)
32	(MSB)	ATTRIBUTE MASK (optional)						
33								
34								
35								
36								
37								
38								
39								(LSB)
40	(MSB)	SESSION ID (optional)						
41								
42								
43								(LSB)

Table 23~~2329~~ - Response to OPEN Object Action

Bit / Byte	7	6	5	4	3	2	1	0
0		SCSI Status Byte						
1	(MSB)							
2								
3								
4								
5		Object_Logical_Length						
6								
7								
8								(LSB)
9	(MSB)							
10		Session ID						
11		(optional)						
12								(LSB)
13	(MSB)							
14								
15		Object Group remaining Capacity						
16								
17								
18								
19								
20								(LSB)

Editor's note: Should the Session ID and Object Group Remaining Capacity be reversed in the order of bytes?

6.5 READ (Mandatory)

The storage device is requested to return data to the ~~Requester~~ application client from a specified object. A priority mechanism to aid the OSD device in reordering queued commands is an option.

Table 24~~2430~~ - Read Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			READ ACTION CODE (8805h)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17								
18								
19				OBJECT ID				
20								
21								
22								
23								(LSB)
24	(MSB)							
25								
26								
27				STARTING BYTE				
28				ADDRESS				
29								
30								
31								(LSB)
32	(MSB)							
33								
34								
35				TRANSFER LENGTH				
36								
37								
38								
39								
40	(MSB)							
41				SESSION ID				
42				(optional)				
43								(LSB)

Data is returned as an information transfer.

Table 2525~~31~~ - Response to Read Object Action

Byte \ Bit	7	6	5	4	3	2	1	0
0				SCSI STATUS BYTE				
1	(MSB)							
2								
3								
4				LENGTH OF DATA RETURNED				
5								
6								
7								
8								(LSB)

6.6 WRITE (Mandatory)

~~This will~~ The WRITE shall cause the specified number of bytes to be written to the designated object at the relative location ~~also~~ specified. Information required is similar to that for a READ. A WRITE to a byte that is greater than the object logical length ~~will~~ shall implicitly increase the logical length of the object. If there is a capacity quota on the Object Group ~~to which~~ for this object ~~belongs~~, the OSD device shall test~~s~~ to make sure the WRITE does not exceed the quota. If it does, the operation is rejected. ~~See {{Annex A1.17}}.~~

Table 2626~~32~~ - WRITE Object Action

Byte \ Bit	7	6	5	4	3	2	1	0
8	(MSB)			WRITE ACTION CODE (8806h)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17								
18								
19								
20				Object ID				
21								
22								
23								(LSB)
24	(MSB)							
25								
26								
27				STARTING BYTE ADDRESS				
28								
29								
30								
31								(LSB)
32	(MSB)							
33								
34								
35				LENGTH				
36								
37								
38								
39								(LSB)
40	(MSB)							
41				SESSION ID (optional)				
42								
43								(LSB)

Data is supplied in an information transfer.

Table 27~~27~~~~33~~ - Response to WRITE Object Action

Bit / Byte	7	6	5	4	3	2	1	0
0	SCSI STATUS BYTE							
1	(MSB)							
2								
3								
4				OBJECT GROUP REMAINING CAPACITY				
5								
6								
7								
8								(LSB)

6.7 APPEND (Mandatory)

~~This will~~ The APPEND shall cause the specified number of bytes to be written to the designated object starting immediately after the object logical length. The information required is similar to that for a READ or WRITE except that no starting location is provided. The OSD device is responsible for determining ~~this~~the start location. The APPEND ~~will~~ shall also cause the logical length of the Object to be updated reflecting the data added by the APPEND command. ~~(This action could easily be constructed as an option on the WRITE action rather than as a separate action.)~~

Data is sent as an information transfer.

Table 28~~2834~~ - APPEND

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			APPEND ACTION CODE (8807h)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17								
18								
19								
20								
21								
22				OBJECT ID				
23								(LSB)
24	(MSB)							
25								
26								
27				TRANSFER LENGTH				
28								
29								
30								
31								(LSB)
32	(MSB)							
23				SESSION ID				
34				(optional)				
35								(LSB)

Table 29~~2935~~ - APPEND to Object Response

Bit / Byte	7	6	5	4	3	2	1	0
0				SCSI STATUS BYTE				
1	(MSB)							
2								
3								
4				STARTING BYTE				
5				ADDRESS				
6				(of APPEND action)				
7								
8								(LSB)
9	(MSB)							
10								
11								
12				OBJECT GROUP				
13				REMAINING CAPACITY				
14								
15								
16								(LSB)

6.8 FLUSH Object (Mandatory)

This ensures all data and attribute bytes for the specified object are stored in non-volatile media.

Table 30~~30~~~~36~~ - FLUSH Object Operation

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			FLUSH OBJECT ACTION CODE (8808h)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17								
18								
19				OBJECT ID				
20								
21								
22								
23								(LSB)

Table 31~~31~~~~37~~ - FLUSH Object Response

Bit / Byte	7	6	5	4	3	2	1	0
0				SCSI STATUS BYTE				

6.9 CLOSE (Optional)

~~This will~~ The CLOSE shall cause the Object to be identified as no longer in use by a given session.

~~Editor's note: John Wilkes asks what happens if optional session ID in OPEN was not given?~~

Table 32~~3238~~ - CLOSE Object Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			CLOSE ACTION CODE (8809h)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17				OBJECT ID				
18								
19								
20								
21								
22								(LSB)
23	(MSB)							
24								
25				SESSION ID				
26								
27								(LSB)

Table 33~~3339~~ - Response to CLOSE Object Action

Bit / Byte	7	6	5	4	3	2	1	0
0				SCSI STATUS BYTE				

6.10 REMOVE (Mandatory)

Deletes an Object.

Table 34~~3440~~ - REMOVE Object Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			REMOVE ACTION CODE (880Ah)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17								
18								
19								
20								
21								
22				OBJECT ID				
23								(LSB)

Table 35~~3541~~ - REMOVE Object Response

Bit / Byte	7	6	5	4	3	2	1	0
0	SCSI STATUS BYTE							
1	(MSB)							
2								
3								
4				OBJECT GROUP				
5				REMAINING CAPACITY				
6								
7								
8								(LSB)

6.11 CREATE OBJECT GROUP (Mandatory)

This command shall cause the OSD device to a~~A~~llocate ~~on the storage device~~ a new set of objects. The operation ~~would implicitly~~ shall establish an object list and group control object for the Object Group.

Table 36~~36~~42 - CREATE OBJECT GROUP Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			CREATE OBJECT GROUP ACTION CODE (880Bh)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14								
15								
16				CAPACITY QUOTA				
17								
18								
19								(LSB)

The response to the CREATE OBJECT GROUP action includes the Object Group ID assigned by the OSD device.

Table 37~~3743~~ - Response to CREATE OBJECT GROUP

Bit / Byte	7	6	5	4	3	2	1	0
0	SCSI STATUS BYTE							

6.12 REMOVE OBJECT GROUP (Mandatory)

This ~~is the~~ function ~~that will~~ shall delete an Object Group from the OSD device. Any Objects ~~it~~ the Object Group contains ~~will~~ shall be deleted.

Editor's note: What is the outcome if an Object is a member of more than one Object Group?

Table ~~3838~~44 - REMOVE OBJECT GROUP Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)	REMOVE OBJECT GROUP ACTION CODE (880Ch)						
9								(LSB)
10	OPTION BYTE 1							
11	OPTION BYTE 2							
12	(MSB)							
13								
14		OBJECT GROUP ID						
15								(LSB)

Table ~~3939~~45 - Response to REMOVE OBJECT GROUP Action

Bit / Byte	7	6	5	4	3	2	1	0
0	SCSI STATUS BYTE							
1	(MSB)							
2								
3								
4		LOGICAL UNIT						
5		REMAINING CAPACITY						
6								
7								
8								(LSB)

6.13 IMPORT Object (optional)

The IMPORT function ~~will~~ shall enable the OSD device to access another OSD device to retrieve a specified object and create another copy of ~~it~~ the object on the requesting OSD device.

This function ~~will~~ shall copy an object from another storage device by issuing OPEN, READs and a CLOSE or just a READ to ~~that~~ the designated ~~storage~~ OSD device, transferring the object. ~~This command effectively issues a CREATE, sufficient WRITEs and a CLOSE to create the object on the addressed OSD.~~

Table 4040~~46~~ - Import Object Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)	IMPORT ACTION CODE (880Dh)						
9								(LSB)
10	OPTION BYTE 1							
11	OPTION BYTE 2							
12	(MSB)							
13		~~DESTINATION~~SOURCE OBJECT GROUP ID						
14								
15								(LSB)
16	(MSB)							
17								
18								
19		SOURCE OBJECT ID						
20								
21								
22								
23								(LSB)
24	(MSB)							
25								
26								
27								
28								
29								
30								
31		SOURCE OSD						
32								
33								
34								
35								
36								
37								
38								
39								(LSB)
40	(MSB)							
41								
42								
43		ATTRIBUTE MASK						
44								
45								
46								
47								(LSB)
48	(MSB)							
49		~~SOURC~~DESTINATION~~E~~ OBJECT GROUP ID						
50								
51								(LSB)

~~**Editor's note: The order of fields in this table has been criticized.**~~

Editor's note: How shall the source ID be parsed? Should it be the first 8 bytes for target and the last 8 bytes for logical unit as in the SPC COPY?

Table 41~~4147~~ - Response to Import Object Action

Byte \ Bit	7	6	5	4	3	2	1	0
0	SCSI STATUS BYTE							
1	(MSB)							
2								
3								
4								
5				OBJECT ID				
6								
7								
8								(LSB)
9	(MSB)							
10								
11								
12				OBJECT GROUP REMAINING CAPACITY				
13								
14								
15								
16								(LSB)

6.14 GET ATTRIBUTES (Mandatory)

The ~~function~~ GET ATTRIBUTES shall retrieve,~~s~~ for the specified object, the ~~meta-data~~attributes associated with the object~~. It is also used to interrogate~~ and the Object Group ~~default and storage device wide~~ attributes. The bit mask identifies the attributes being interrogated. An inbound (OSD to ~~requester~~application client) information transfer transmits the attributes to the ~~requester~~application client. Security keys are not returned.

Table 42~~42~~~~48~~ - GET ATTRIBUTES Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			GET ATTRIBUTES ACTION CODE (880Eh)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17								
18								
19				OBJECT ID				
20								
21								
22								
23								(LSB)
24	(MSB)							
25								
26								
27				ATTRIBUTE MASK				
28								
29								
30								
31								(LSB)
32	(MSB)							
33				SESSION ID				
34				(optional)				
35								(LSB)

Table 43~~43~~~~49~~ – Response to GET ATTRIBUTE Action

Bit / Byte	7	6	5	4	3	2	1	0
0				SCSI Status Byte				

Editor's note: The Attribute Mask needs to be specified.

6.15 SET ATTRIBUTES (Mandatory)

The SET ATTRIBUTES function shall sets~~s~~ attributes for a specified Object. The attributes values are sent via an outbound (from ~~requester~~application client to OSD device) information transfer. The Security keys (See 5.4.2.1~~5.3.3.1~~ for a description of these attributes) ~~are~~ shall be the only attributes that are set by the SET ATTRIBUTE action but cannot be read by the GET ATTRIBUTE action. If the session ID field is non-zero, then the attributes being set shall apply to the session and not to the ~~(static)~~ object.

Editor's note: Does the latter requirement suggest that the if the session ID is non-zero, the Object ID should be zero?

Table 4444~~50~~ - SET ATTRIBUTES Action

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SET ATTRIBUTES ACTION CODE (880Fh)				
9								(LSB)
10				OPTION BYTE 1				
11				OPTION BYTE 2				
12	(MSB)							
13								
14				OBJECT GROUP ID				
15								(LSB)
16	(MSB)							
17								
18								
19				OBJECT ID				
20								
21								
22								
23								(LSB)
24	(MSB)							
25				SESSION ID				
26				(optional)				
27								(LSB)
28	(MSB)							
29								
30								
31								
32				ATTRIBUTE MASK				
33								
34								
35								(LSB)

Table 4545~~51~~ - Response to Set Attribute Action

Bit / Byte	7	6	5	4	3	2	1	0
0				SCSI Status Byte				

Annex A Research Notes (informative)

A.1 Overview

~~This section~~ Annex A 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.

Editor's note: This annex should be deleted or set in a different vane prior to forwarding OSD

A.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 ~~SET ATTRIBUTE command could turn the OSD device operation off~~ a vendor specific action (e.g., download microcode) needs to be taken. This ~~should~~ may cause the entire contents of the storage device to be destroyed.

A.3 CREATE

Possible options:

Bit / Byte 0	7	6	5	4	3	2	1	0
	reserved	Reserved	reserved	reserved	reserved	ATTR	SESS	CMPL

Figure 1 CREATE Action Option byte 2

CMPL is set to one when the CREATE action is to cause a complete object to be created, written and closed. A data phase associated with the CREATE action will convey the object data to the OSD device. The LENGTH field in the CREATE Action will contain the length of this data. This is intended to improve performance in environments where many small objects are being created.

SESS is set to one if the ~~requester~~application client requires the OSD device to set up an I/O session. This will require the OSD device to maintain state for the duration of the OPEN session. ~~It~~ SESS is ~~made an~~ optional because it was thought that many or most I/O would not have quality of service requirements attached to them. In this case no session state is maintained.

ATTR is set to one to indicate that the information transfer includes attribute data for the object to be created.

The CREATE 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 ~~shall~~ 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 ~~. In other words,~~

~~have~~such that the ~~requester~~application client describes the requirement rather than the solution.

Some of the object attributes discussed, in addition to those in 5.3, include:

Make this file password protected. Rejected as not the right place to put a password.

- 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.
- Specify if sub object level locking is required. This is still ~~too ill-defined~~not fleshed out.
- Specify versioning. This was rejected – as more appropriately done by the OS.
- Mirror support - cause all updates to be mirrored onto another object. No one ~~could~~ has come up with a concrete requirement for this.
- Allocate space in units of a specified minimum size. The motive for this attribute was to allow the ~~requester~~application client to specify a minimum degree of contiguity as mentioned above.
- Set rights (as in UNIX)
- Create an object to emulate a~~n~~ ~~BBSD~~SBC ~~SBC~~ storage device. Assign a LUN to it. (LUN ~~shall be~~ returned upon completion.)

A.4 OPEN

Possible Options:

Bit Byte 0	7	6	5	4	3	2	1	0
	reserved	reserved	reserved	RDNLY	WRNLY	ATTR	SESS	SEQ

Figure 2 OPEN Action Options

SEQ is set to indicate that this session ~~will~~ should access the object sequentially.

SESS is set to one if the ~~requester~~application client requires the OSD device to set up an I/O session. This ~~will~~ may require the OSD device to maintain state for the duration of the open 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 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 session that is to consist of WRITEs only.

RDNLY is set to one, the OSD device is instructed to only allow Reads to the referenced object. WRONLY and RDNLY cannot both be set for a given session.

There was much discussion on the merits of an OPEN 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 ~~can~~ may be expressed to the OSD device. The storage device ~~can~~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 ~~could~~ may be implicit with the first READ on an object.

One view of the OPEN and CLOSE commands is that they ~~can~~ should frame the usage of an object. The storage device ~~could~~ may use the awareness of an object not being open as the signal that it ~~can~~ 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 ~~could~~ 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 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 ~~requester~~application client ~~could~~ 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 session ID ~~could~~ may let the OSD device discriminate between multiple sets of quality of service requests.

~~Were~~ If SESSION ID's were always used, there would be no need for both a SESSION ID and the [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 ~~could~~ 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 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 ~~requester~~application client ~~can~~ may simply READ an object without first issuing an OPEN (again, assuming no QoS requirements).

The optional length parameter for the OPEN allows an ~~Requester~~ 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 session and might be specified with the size attributes in 5.3 rather than an explicit argument on this action. The ~~requester~~application client ~~can~~ 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 ~~requester~~ application client ~~(client or host, if you like)~~ caching.

A.5 READ

Possible options:

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved	reserved	reserved	reserved	PRIORITY			

Figure 3 WRITE Action Option byte 2

PRIORITY is a 4 bit integer that puts a relative time criticality on the submitted request. A lower value is a higher priority. The OSD device ~~could~~ may use this to help order the execution of the requests in its queue. The intent of priority is to identify classes of relative performance in the I/O queue. A system ~~could~~may, for instance, decide to define class 4 as normal I/O, with class 5 being background work and class 3, exceptional request that ~~shall~~ should be put ahead of all normal requests.

It ~~could~~ has been argued ~~(and was unendingly!)~~ 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 ~~can~~ may return the entire contents of an object by supplying a length longer than that of the object. The OSD device ~~will~~ should return all data and, in the response, the actual length of the data sent to the ~~requester~~application client. For instance, a READ with a length of 64K ~~can~~ may be used to read any object having a length less than or equal to 64K. The READ ~~will~~ should send back as much data as the object contains, with the length of that data supplied in the response.

A.6 WRITE

Possible options:

Bit / Byte	7	6	5	4	3	2	1	0
0	reserved	reserved	reserved	Reserved	PRIORITY			

Figure 4 WRITE Action Option byte 2

An attribute on the Object to which the WRITE has been issued ~~could~~ 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 ~~could~~ may automatically cause the WRITE to be propagated to another OSD to keep it in sync with the written to OSD.

A.7 APPEND

Possible options:

Bit / Byte	7	6	5	4	3	2	1	0
0	reserved	reserved	reserved	Reserved	PRIORITY			

Figure 5 APPEND Action Option byte 2

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 ~~requesters~~ application clients ~~could~~ 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 ~~could~~ just issues an ~~WRITE~~APPEND. This feature does not work in all cases, of course, but logging and similar operations ~~can~~ may be supported where the precise ordering of the log entries is not a concern.

A.8 FLUSH Object

This command ~~can~~ may also be used for synchronizing the object group or the storage device by specifying the appropriate value to indicate that one or all Object Groups are to be flushed.

A.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 ~~Requester~~ application client any longer. The OSD device ~~could~~ 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, ~~could~~ 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 ~~could~~ may either discard the partially created object or it ~~could~~ may establish the existence with the data that has been written to it so far. Which of these to actions to take ~~could~~ may be a matter of policy, established at the OSD device, Object Group or object level.

Some believe that ~~Requester~~ application client failure recovery is made harder by requiring a CLOSE in order to not lose written data, especially as the FLUSH operations ~~can~~ 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 actions, a CLOSE action on an object is an assertion that ~~Requester~~ application client activity with this object has ceased.

A.10 REMOVE Object

Potential~~ssible~~ options:

Bit / Byte 0	7	6	5	4	3	2	1	0
Byte 0	reserved	reserved	reserved	reserved	reserved	reserved	reserved	DESTR

Figure 6 REMOVE Object Action Option byte 2

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 ~~must~~ needs to be resolved is the issuance of a REMOVE 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 actions.

A.11 CREATE OBJECT GROUP and REMOVE OBJECT GROUP

There was considerable disagreement on the need for Object Groups, 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 OSs because it is quite likely that different file systems, databases, volume managers and virtual memory systems ~~will~~ may not be coded to use distributed capacity allocation protocol among themselves. The operation supporting Object Groups 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 Object Groups 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 ~~can~~ 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 ~~can~~ may be oversubscribed or not.

This function will also create a well-known object holding Object Group attributes, which ~~cannot~~ should not be removed as long as the Object Group exists. ~~This~~ The root object ~~will~~ may serve as the starting point for navigating the objects in the Object Group.

The 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.

A.12 IMPORT Object

Bit / Byte	7	6	5	4	3	2	1	0
Byte 0	reserved	reserved	reserved	reserved	PRIORITY			

Figure 7 APPEND Action Option byte 2

It was not unequivocal that IMPORT 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.

A.13 GET OBJECT 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 ~~requester~~application client and the OSD device, For example, 5.3 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 ~~could~~ may return the storage device Object and all Object 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 ~~could~~ 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.

A.14 SET OBJECT 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 ~~will~~ may want to ensure that names are not ambiguous, the thought is that setting the name should be a closely controlled operation.

A.15 GET, SET STORAGE DEVICE ASSOCIATIONS

Though not included in the commands, a method for defining and interrogating sets of OSD ~~could~~ may be useful, especially in support of RAID configurations.

These actions ~~would~~ may define or interrogate relationships among storage devices. This ~~would~~ may be necessary for inter-storage device communications. (A possible implementation ~~would~~ 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 ~~would~~ may be responsible for disseminating to the other members changes in set attributes. Other members ~~would~~ 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 ~~could~~ 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.

A.16 Sessions

Sessions are sets of I/O requests that are to honored by the OSD device in the same way. That is, the ~~Requester~~ application client ~~can~~ may use the Set Attribute action to establish the quality of service characteristics associated with the session. The OSD device is expected to accept the contract – by virtue of not rejecting the Set Attribute action with session attributes – and process the I/O requests per those quality of service requirements. The session ID is returned by the OSD device in the response to an OPEN or CREATE action. The ~~Requester~~ application client then supplies this session ID in each Read, WRITE, or APPEND request. In addition the ~~Requester~~ application client ~~can~~ may supply the ID of a previously set up 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 session was originally established.

A.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 ~~spec~~ 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 ~~spec~~ standard ~~could~~ may certainly feel free to provide additional guarantees.

IMPORT should be modified to move a range of bytes within an object rather than whole objects. This ~~can~~ may be done by adding two offsets and a length to the description of an IMPORT 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 ~~could~~ may perform a restriping operation from an m disk array to an n disk array with at most n*m total IMPORT directives without forcing multiple reads or writes of the same bytes.

A.18 Attributes

The distinction between Object logical Length and Object Size is that the former defines the range of addresses in which read 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 action on the object.

A.19 Policy

Policy refers to the set of conditions and subsequent actions (to be performed in the event the associated condition~~(s)~~ 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, ~~and~~ 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 ~~document~~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 ~~requesters~~ application client's good guidance for laying out objects and transferring on wise boundaries. This ~~can~~ 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.

A.20 Rationale 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:

- Reduced device dependencies – the attributes contain no device specifics.~~;~~ ~~indeed,~~ A~~a~~n OSD device is free to self-manage and optimize itself as long as the result continues to achieve the attribute specifications.
- Higher level of specification – the attributes are biased in the direction of processing and/or data requirements, rather than device capabilities.
- Function enablement – functions such as data movement, querying, and performance optimization that are often performed at the host ~~can~~ may be performed elsewhere in the network (e.g., by a storage management processor) or by the OSD device~~ itself;~~. A~~a~~dditionally, 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 B OSD related Topics (informative)

B.1 Relationship to file systems

Data is stored in fully self-defined and self-contained objects~~, which~~that the storage device ~~on which they reside~~ is responsible for maintaining. Within the constraints of the security policy~~ in place~~, the object-based architecture makes it possible for any ~~requester~~application client to inspect a storage device to discover what objects exist there and access them. The application client ~~It~~ 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 ~~(I~~it ~~could~~ may ~~reasonably~~ be ~~argued~~ that the OSD-Object Group-object organization is, in fact, a hierarchy.~~)~~ There is no naming function. An object ~~might~~may contain several files, or an object may only be part of a file. It is ~~hoped~~ assumed that Objects ~~can~~ may be operating system independent, with ~~(~~almost~~)~~ any OS able to superimpose its file system structures on the object abstraction. The OS files, directories and ~~meta-data~~attributes, ~~–~~ other than space management mechanisms, ~~–~~ are constructed as objects.

The object abstraction ~~shall~~ 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 ~~to do so~~ without infringing on its ~~sovereign~~ 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 ~~could~~ may take the form of service levels and capacity quotas for each.

Reliability characteristics ~~would~~ are also be attributes. A disc array ~~could~~ may include mirror, RAID and JBOD storage. An object ~~could~~ may be created on the type of storage that provides the appropriate~~d~~ reliability level as a result of a reliability attribute supplied with a CREATE action.

B.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) ~~can~~ may use the same OSD without conflict. ~~, since they usually assume that others do not exist.~~

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 ~~can~~ 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 ~~could~~ 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 ~~is~~ 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 ~~could~~ may contain objects in the high data rate zones (~~possibly~~ 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~~, of course~~. A video processing shop might have several projects underway simultaneously. The objects for each project ~~could~~ 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 ~~will~~ maintains an object list for each of its Object Groups and space management information for the entire storage device.

B.3 Identifiers (ID's)

The identifier associated with an object is chosen by the storage device and returned to the ~~Requester~~ application client in response to the command to create an object. The ID will be an unsigned 64-bit integer. ~~The length could be set to a smaller size by defining a storage device attribute that directs the OSD device to only issue ID values less than 64 bits – perhaps 32 bits.~~ Specific ID's ~~could~~ may be reserved for well-known objects. ~~Experience to date suggests some w~~Well known objects ~~will~~ are needed to enable a file system to start navigating through the objects on the OSD device.

~~There were a couple of suggestions for~~ Allowing the ~~Requester~~ application client to supply OBJECT ID at CREATE time, instead of having the OSD device assign and return it~~. This makes~~ may have made garbage collection easier for the ~~Requester~~ application client, but would have introduced~~s~~ 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 ~~left with~~ specified that the OSD device assigns~~ing~~ it~~, with the recognition that it is a subject needing further investigation~~.

B.4 Relationship to Sector Based Storage devices

B.4.1 Emulating a ~~BBSD~~SBC ~~-~~ device on an OSD device

While there will likely be a continuing need to support ~~BBSD~~SBC and relative sector addressing, there is a security problem with allowing a storage device to switch from OSD to ~~BBSD~~SBC and back. To ensure data security the change from ~~BBSD~~SBC to OSD ~~shall -~~ and the change from OSD to ~~BBSD~~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 ~~BBSD~~SBC storage device on an OSD device. The OSD device ~~can~~ may allocate an object matching the desired size of the logical ~~BBSD~~SBC. It assigns this a LUN, letting an ~~Requester~~ application client that cannot support the ~~OBS~~OSA build its file system and access data in this special object as though it were a separate physical storage device.

Editor's note: This capability may not survive the development.

B.4.2 ~~BBSD~~SBC SCSI commands

The response to the following SCSI commands would be invalid operations in OSD mode:

FORMAT UNIT	SEEK	READ EXTENDED	VERIFY	REBUILD
REASSIGN BLOCKS	RESERVE	WRITE EXTENDED	READ LONG	XDWRITE
RELEASE	READ	SEEK EXTENDED	WRITE LONG	XPWRITE
READ CAPACITY	WRITE	WRITE AND VERIFY	WRITE SAME	XDREAD
READ DEFECT	REGENERATE		PRE-FETCH	

~~The following SCSI commands would be valid on an OSD device:~~

~~MODE SELECT~~	~~PREVENT/ALLOW MEDIUM REMOVAL~~	~~INQUIRY~~
~~MODE SENSE~~	~~SYNCHRONIZE CACHE~~	~~REQUEST SENSE~~
~~START & STOP~~	~~TEST UNIT READY~~	~~SEND DIAGNOSTIC~~

B.5 Data Sharing and Concurrent Update

~~Some who have participated in the NASD research think that s~~Scalability ~~could~~ may be improved with an optimistic control scheme where ~~Requesters~~ application clients ~~can~~ may act as though there is no conflict unless there are actually simultaneous attempts to access the same records by multiple ~~Requesters~~ application clients. They ~~could~~ may learn this from the OSD device ~~themselves~~ when attempting to gain control of objects for the purpose of updating them. An attribute set by an ~~Requester~~ application client establishes that it intends to update certain data. The attribute is reset after completion of the WRITEs. Only if another ~~Requester~~ 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 ~~today~~ 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 5.4.3.1~~5.3.4.1~~ ~~can~~ 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 ~~shall~~ evaluates successfully before the rest of the action is enacted, then attribute values ~~can~~ may be tested and modified and action may~~can~~ be conditional. In this scenario, the semantics of Clause 5.4.3.1~~5.3.4.1~~ 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).

~~The feedback from data base community has consistently been that they do not want any help in this area. That is fine; anyone with a concurrency control technique can continue to use it. Any new work in this area would only be another option that could be used if found beneficial. It is expected that more work will be done in this area.~~

B.6 OSD and aggregation

B.6.1 Overview

Aggregation ~~is used here to~~ 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 ~~shall be~~ 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. ~~There shall be a means~~

~~by which a~~ A file system may~~can~~ create, access and manage a collection of objects located on several storage devices as if they constituted a single file.

When an ~~Requester~~ application client solicits permission to access a file that is held in multiple objects, it ~~shall~~ should also have mapping information that will let the ~~Requester~~ application client direct its I/O commands to the appropriate storage device. It is transparent to OSD devices and the objects they contain ~~are assumed to be unaware~~ that they may be only parts of larger storage or data constructs. (~~The still un~~If defined, Storage device Association ~~being the~~ is an exception.) This does not preclude an OSD device subsystem from internally aggregating among OSD ~~device's~~logical units it manages.

B.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 ~~can~~ may be adapted to work on the object interface. ~~In fact, it is possible for the~~ XOR operations may~~to~~ be done implicitly whenever a WRITE command addresses an OSD device that is parity protected. The OSD device ~~can~~ 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 ~~could~~ 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 ~~requester~~application client.

The ~~requester~~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 ~~requesters~~ application client can be responsible for e~~i~~nsuring the WRITEs are issued to both storage devices, or the mirror storage devices ~~can~~ 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 ~~could~~ may be ~~possible~~ used as each method has its advantage.

B.6.3 Aggregation for capacity - spanning

As ~~was~~ described above, for databases and other large files that ~~can~~do not fit on a single storage device, there ~~shall be~~ is a method for spreading them across several devices. This ~~is~~ has been done ~~today~~ in legacy systems by creating logical volumes that span disc drives. When an ~~Requester~~ application client ~~wishes~~ 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 ~~OBS~~OSA ~~would~~ uses a similar method.

The ~~Requester~~ application client needs a mapping function similar to that of the volume manager in a ~~BBSD~~SBC environment. When the ~~requester~~application client ~~goes to~~ accesses the Policy/Storage Manager to get authorization to access the database, the spanning information is also returned. The ~~Requester~~ application client ~~will~~ may use this to transform a file request into an object request to the appropriate OSD.

B.6.4 Aggregation for performance – striping

Striping is handled in the same way as spanning, except that a single large-data request ~~will~~ results in an object request to each of the storage devices in the stripe group.

B.6.5 Accessing Aggregated Objects

The following is a description of one method for accessing aggregated objects. ~~It is included to illustrate one method for supporting aggregated objects. It is by no means the only possibility.~~ It assumes an option in Option byte 1:

Bit / Byte	7	6	5	4	3	2	1	0
Byte 0	Reserved	Reserved	Reserved	DPO	FUA	Reserved	Reserved	NO-REDIRECT

Figure 8 Option byte 1 support for aggregation

~~When~~ NO-REDIRECT ~~is~~ set to one ~~,~~indicates that the ~~Requester~~ application client does not support mapping of an object onto multiple OSD and ~~will~~ does not accept mapping information.

When a Requestor accesses an object, it ~~shall~~ 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~~nother is that~~ it ~~might~~ may negotiate with an external service (such as a file system policy manager) to obtain this ~~capability~~information. The ways in which an ~~Requestor~~ application client might perform such a negotiation are beyond the scope of this ~~document~~standard. ~~, but in this discussion it is assumed that a Policy/Storage Manager is supplying capabilities.~~

The essence of this suggested method for accessing aggregated objects is that an ~~Requester~~ 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 C.6.4~~0~~. 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.

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 ~~Requesters~~ application clients be able to directly access individual OSD devices providing backing store for virtual (aggregated) objects. To do this, ~~Requesters~~ application client ~~must~~ need to become aware that the object they are addressing as {OSD-NAME, OBJECT GROUP ID, OBJECT ID} is in fact mapped over for example {{OSD-NAME1, OBJECT GROUP ID1, OBJECT ID1}, {OSD-NAME2, OBJECT GROUP ID2, OBJECT ID2}, …}~~ for example~~.

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~~ or not~~. It may also choose to act as a proxy on behalf of the requestor~~ or not~~. If the NO-REDIRECT bit is set in OPTION BYTE 1 of the request, the Storage Manager ~~shall~~ acts as a proxy on behalf of the Requestor, and may not include the mapping of the object onto the aggregate members.

B.6.6 Description of Aggregate Layouts

~~In this subsection a possible structure for virtual object maps is described.~~ To ensure the ability of all ~~Requesters~~ application clients to be able to at least throw away layout descriptions that they ~~cannot~~ are not able to parse, layout descriptions ~~shall~~ 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 ~~shall be~~ is a layout type field (1 byte). If the layout type is ~~unfamiliar~~not recognized, the ~~Requester~~ application client ~~can~~ 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:

1 An Obj-Descrip is a tuple containing {OSD-NAME, OBJECT GROUP ID, OBJECT ID, SIGNED CAPABILITY}, where Signed Capabilities are described in C.2.3,
2 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.

The following are defined for type-specific layout descriptions:

3 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 ~~"trivial"~~ storage management option of a non-mirrored, non-striped object by describing a mirrored set with only one member.
4 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 ~~possibly~~ may hav~~ing~~e different layouts.
5 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.
6 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~~ (cite Edward Lee's ACM Transactions on Computers paper on parity distributions)~~. The rest of the bits contain a list, of length specified by the first field, of fully-qualified layout descriptions.
7 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 ~~Requester~~ application client to obtain attributes of each constituent object before issuing re-directed object accesses.
8 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 ~~Requester~~ application client software with specialized layout algorithms.

Note 4 :~~Note that m~~Most layout description types ~~can~~ 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. ~~It is also possible that i~~Individual vendors might define non-nested layout types, although this is not recommended.

B.6.7 Other Type Values

Other layout type values that ~~should possibly~~ 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 ~~(cite Peter Chen's 1994 ACM Computing Survey's article)~~.

B.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 46 - Aggregation: Simple Mirroring Descriptor

Value	Size (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

~~Figure 9 Aggregation: Simple Mirroring Descriptor~~

B.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 47 - Aggregation: Striping Descriptor

Value	Size (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
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)

~~Figure 10 Aggregation: Striping Descriptor~~

B.6.10 Storage Managers Responding to Requests

There are four possibilities for what an ~~Requestor~~ 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 ~~Requester~~ application client ~~shall~~ allocates at least 4 bytes more than the expected response size to be able to determine if a valid map was received (because an ~~Requester~~ application client does not necessarily know the size of the included map).

Table 48 - 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-REDIRECT is specified in the request.
2	Y	N	The Storage Manager responds with only the mapping of the object onto the aggregate members. The ~~Requester~~ application client ~~shall~~ repeats the request, either by mapping the request onto the members of the aggregate, or back to the Storage Manager after setting the NO-REDIRECT bit in Option Byte 1. This ~~is~~ should be disallowed when NO-REDIRECT is specified.
3	N	Y	The Storage Manager transparently completes the request on behalf of the ~~Requester~~ application client, and forwards the result to the ~~Requester~~ application client. To the ~~Requester~~ 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).

~~**Figure 11 Aggregation: Responses**~~

When NO-REDIRECT is specified for a request, response type=3 is the only valid successful result of an operation.

B.6.11 Example 1

In this example, an ~~Requester~~ 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 ~~possibly~~ separate Policy/Storage Manager that may not know the mapping for the virtual object. The Storage Manager ~~which~~ 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 ~~Requester~~ 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 49 - Example 1: Read Aggregated Object

Step	Operation	Key Attributes	Source	Destination	Options
1	Get Attribute	V_obj_id	~~Requester~~Application client	Storage Manager	-
2	Get Attribute response	Attributes, map	Storage Manager	~~Requester~~Application client	Response, map included
3	Read	obj_id-1	~~Requester~~Application client	OSD1	-
4	Read	obj_id-2	~~Requester~~Application client	OSD2	-
5	Read	obj_id-3	~~Requester~~Application client	OSD3	-
6	Read response	data	OSD1	~~Requester~~Application client	-
7	Read response	data	OSD2	~~Requester~~Application client	-
8	Read response	data	OSD3	~~Requester~~Application client	-

~~Figure 12 Example 1: Read Aggregated Object~~

If the Requestor did not ~~understand~~ recognize the map provided in Step 2, or is not able to implement the layout specified by the map, ~~one might see~~the result might be:

Table 50 - Example 1: Read Aggregated Objects without mapping support

Step	Operation	Key Attributes	Source	Destination	Options
1	Get Attribute	V_obj_id	~~Requester~~Application client	Storage Manager	-
2	Get Attribute response	attributes, map	Storage Manager	~~Requester~~Application client	response, map included
3	Read	V_obj_id	~~Requester~~Application client	Storage Manager	NO-REDIRECT
4	Read response	data	Storage Manager	~~Requester~~Application client	-

~~Figure 13 Example 1: Read Aggregated Objects without mapping support~~

B.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 ~~chooses to~~ does not provide any data with the map it returns on the first read of the virtual object.

Table 51 - Example 2: Read Aggregated Objects

Step	Operation	Key Attributes	Source	Destination	Options
1	Read	V_obj_id	~~Requester~~Application client	Storage Manager	-
2	Read response	map	Storage Manager	~~Requester~~Application client	map included
3	Read	obj_id-1	~~Requester~~Application client	OSD1	-
4	Read	obj_id-2	~~Requester~~Application client	OSD2	-
5	Read	Obj_id3	~~Requester~~Application client	OSD3	-
6	Read response	data	OSD1	~~Requester~~Application client	-
7	Read response	data	OSD2	~~Requester~~Application client	-
8	Read response	data	OSD3	~~Requester~~Application client	-

~~Figure 14 Example 2: Read Aggregated Objects~~

In this example, if the ~~Requester~~ application client ~~could~~ does not perform the mapping~~ itself~~, ~~this would look like~~the results are:

Table 52 - Example 2: Read Aggregated Objects without mapping support

Step	Operation	Key Attributes	Source	Destination	Options
1	Read REQ	V_obj_id	~~Requester~~Application client	Storage Manager	-
2	Read REP	map	Storage Manager	~~Requester~~Application client	map included
3	Read REQ	V_obj_id	~~Requester~~Application client	Storage Manager	NO-REDIRECT
2	Read REP	Data	Storage Manager	~~Requester~~Application client	Response incl.

~~Figure 15 Example 2: Read Aggregated Objects without mapping support~~

B.7 Booting From an OSD device

~~Initial loading of the OS and starting configuration is possible with OSD. In fact one of the OBSOSA goals is to make that process even more efficient.~~

B.7.1 Cold Boot

This sequence should not be much different from a sequence on ~~BBSD~~SBC. The same files ~~will have~~need to be located and read. The page or swap files ~~will~~ 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’s Group Control Object, instead of fixed sector addresses used by ~~BBSD~~SBCs.

B.7.2 Warm Boot

The warm boot process exposes a fundamental conflict between the desires of minimized boot up time versus full security. OSD ~~can~~ may provide several choices on how to achieve balance ~~or compromise~~ among the choices.

During the shutdown of some systems, it ~~could~~ may be fairly easy for a system to define a quick boot image object and write the required state to it. This ~~could~~ 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 ~~could~~ may read this object to recover its running state and be up and going ~~quite~~ quickly. The ~~requester~~application client writing the boot image ~~could~~ 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 ~~could~~ may reveal, to the system starting up, the list of boot storage devices in some priority order. The low level code in the system ~~could~~ may go down this list to find a valid, well-known boot object. The access control on this object ~~could~~ may be a special case allowing ~~for anonymous~~ READs~~. That is, it could be read~~ without the need for authentication~~ (which admittedly has not been defined yet!)~~. It is conceivable that a denial of service attack ~~could~~ may be used on this object. Since anyone can read it with no special permission, an intruder ~~could~~ may continuously read to disrupt the other systems that have a valid requirement to do so. ~~Some solutions were proposed, but nothing seems to quite fill the bill yet.~~

Note : ~~Note that t~~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 ~~could~~ may just as readily read it. Also the recorded state may include several open files. The object attributes ~~could~~ 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 ~~shall~~ should be done.

B.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 ~~could~~ may be built using their capabilities. ~~The NSIC/NASD working group believes that OSD can result in the benefits enumerated in Section 0.~~ However, other I/O subsystem components ~~must~~ 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 ~~that the NSIC believes to be~~ necessary in order for consumers to realize the benefits of OSD.

Table 53 - OSD Dependencies

Component	What's 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~~A~~). VI~~A would~~ allow s file systems and database managers to communicate directly with OSD devices without traversing the system I/O or network stack once a connection ~~was~~ is established. The low latency achieved thereby ~~would~~ 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 ~~would~~ requires modification of the data managers to use a VI~~A~~ interface, in addition to the modifications listed above.
I/O software stack	OSD devices require commands and responses that are not part of ~~typical~~ legacy storage device driver stacks ~~today~~. Drivers ~~would~~ have to evolve to issue OSD commands and respond appropriately. APIs ~~would~~ 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 ~~can~~ 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) ~~will~~ have to be modified to take advantage of those features. In general, the change required ~~to~~ 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.

~~**Figure 16 OSD Dependencies**~~

Annex C
Known Unresolved Issues or Uncompleted Topics (informative)

C.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 ~~OBS~~OSA activity to ~~itself~~ OSD devices ~~could~~ may pose capacity, throughput and fault tolerance problems. These ~~could~~ 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 ~~non-magnetic NVRAM~~ 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 ~~obviously~~ reduce the application bandwidth.

~~It might seem that a single extra message for each I/O is not a big deal, but i~~In a transaction environment a single extra message for each I/O ~~it is a BIG deal!~~ significantly degrades performance. All I/O is essentially a message and the I/O subsystem is typically message bound. ~~We already have some experience with this kind of thing on a drive where the SMART feature causes logging of environmental data. It can have a significant impact on performance. And the SMART log data is only summary information.~~

C.2 Clocks

Each storage device ~~will~~ should have a readable monotonically incrementing clock to be used for time stamping secure messages and object attributes. Either this clock ~~must~~should ~~-~~be synchronized securely system wide, or the server will ~~have~~ need to accommodate the discrepancies in values from storage device to storage device. The system will ~~have~~ need to provide a mechanism to reload the clock, or the server's~~'~~ accommodation, should the storage device lose power.

~~It is possible that t~~Time stamps on object attributes ~~will~~ may not have the accuracy that the same stamps would have if they were constructed by a server-based Policy/Storage Manager software or ~~Requesters~~ application clients. Such server systems ~~can~~ may be accommodated by ~~OBS~~OSA because the server software ~~can~~ may construct their own time stamps and record them in the FS-specific attribute space (~~which is~~ uninterpreted by the OSD device). However, very inaccurate (low resolution) OSD clocks ~~will~~ 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 ~~guarantee~~ 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.

C.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 ~~OBS~~OSA objects~~, but this will have to be added~~. 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 ~~have~~ need to include a means for getting the status of all sessions by SESSION ID.

C.4 Responses

A single byte has been allocated to hold the standard SCSI response. ~~It may well be that this is insufficient;~~ But the OSD device may have need of more error reporting capability than is possible in a single byte. ~~This should be looked at but has not yet been adequately investigated.~~

C.5 Addressing

The limits on node addressing ~~is seen as~~ may be too restrictive. In addition, the fixed association of ~~Requester~~ application client with a single network address is too restricting. An ~~Requester~~ ~~application client~~ host system ~~could~~ may easily have two or more NIC~~'~~s. It should be possible to take advantage of this without explicit direction from the ~~Requester~~ application client to change from using one address to using another.

Editor's note: Why isn't this transparent to OSD?

C.6 Security

C.6.1 Overview

There are several schools of thought concerning data security. One view is that we ~~can~~should not depend on the entire network being secure.~~,~~ ~~so s~~Some type of cryptographic protocol is needed to build data security upon. (The distribution of Policy/Storage Manager keys to ~~Requesters~~ application clients and OSD devices throughout the addressable network requires some secure distribution method outside the scope of this ~~proposal~~standard.)

Signing and encrypting data at the application before sending it to ~~OBS~~OSA systems solves one part of the problem. It prevents unauthorized agents ~~which~~ that are able to snoop messages on the interconnect from gathering much useful information from the transmissions. In this scheme the storage device (OSD or ~~BBSD~~SBC) is unaware of whether the data is encrypted or not and simply stores it or retrieves it as requested. This doesn't require any new features in the storage. However, this approach does not prevent an unauthorized agent from sabotaging ~~(vandalizing?)~~ data as it is transferred so that later access is not able to recover the appropriate data and it does not prevent unauthorized agents from consuming storage capacity or deleting ~~(~~and overwriting~~)~~ previously stored~~age~~ data.

C.6.2 Encryption Considerations in the long CDB format

To protect ~~OBS~~OSA resources, in addition to user data, commands ~~shall~~ should be verified by an OSD device to be authentic and unaltered. The inclusion of a digital signature in the action-specific fields or the inclusion of a weaker verification code (such as CRC) in the action-specific fields of an encrypted command provides the needed verifiability. If encryption is ~~(~~also~~)~~ employed, then data privacy is also provided.

The ~~ANSI-approved~~ SPC-2 long CDB format has provision for indicating that cryptographic operations ~~shall~~ should be executed by a receiving OSD device before it accepts a command.

~~The encryption identification field in the long CDB format provides this indication.~~ The first 8 bytes of the CDB are never encrypted. ~~-~~When the encryption identification field (~~which~~ that is within the 8 unencrypted bytes) indicates that the CDB is encrypted, the storage device ~~shall~~ applies~~y~~ cryptography to the rest of the CDB bytes. The action-specific fields should include CRC, digital signature or other verification bytes so the target OSD ~~can~~ may verify that the command has not been modified in transfer.

When the CDB is encrypted, it may be important that all commands be the same size. If they aren't, an unauthorized entity ~~could~~ may learn information about the commands based on the transferred sizes. This is where the filler bytes come in. Each command ~~will~~ should have enough filler bytes added so that all encrypted commands will be a standard size.

The remain~~ing sections~~der of this c~~C~~lause ~~and all of the next~~ describe one strategy for defining the action-specific payload or data content using this long CDB. ~~The security scheme described was developed for OBS~~OSA ~~based on experience from the CMU NASD implementation.~~

In the long term, ~~the CMU team assert, s~~Security ~~shall~~ should be ensured by mechanisms that protect the integrity and privacy of OSD communications. To avoid specialization to any particular file system or server application, OSD security mechanisms ~~shall~~ should correctly enforce a wide range of possible access policies whose details ~~will~~ 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 and user authentication/authorization are beyond the scope of this ~~proposal~~ standard. ~~Instead, t~~This ~~proposal~~ annex defines mechanisms for an OSD device to authenticate that a command has been authorized by a Policy/Storage Manager and for encrypting and decrypting these commands according to Policy/Storage Manager specifications.

Authentication is implemented by utilizing a secure hash function to provide digital signatures on requests, responses, and capabilities. Privacy is implemented by using an encryption function to permute action and data bits to ensure that an eavesdropper who does not possess a secret shared by both communication endpoints cannot interpret the transferred bits.

The keyed secure hash function in this ~~proposal~~ annex is defined to be HMAC-SHA-1, and the encryption function is Triple-DES.

Security is achieved through the sharing of secrets. Fundamentally, to perform an access, an ~~Requester~~ application client ~~shall~~ demonstrates that either it shares a secret with the OSD device, or that the access was previously authorized by a Storage/Policy Manager~~, which~~that shares a secret with the OSD device. Key management is a critical security function, but as it is inherently a system-wide function, it is properly a function of a higher level than the storage system. ~~For our purposes, k~~Keys ~~shall~~ should be distributed privately to ~~Requesters~~ application clients by Storage/Policy 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.

C.6.3 Secrets / OSD Key Hierarchy

The secrets shared between OSD devices and other members of the distributed storage system are known as ~~K~~keys. Keys are 128-bit values. Keys ~~shall~~ should be transmitted from time to time, so any operation transmitting a key ~~shall~~should be encrypted with a different key. Because each use of a key exposes that key to a degree of attack, keys that are used regularly should be changed regularly. To facilitate this rotation of keys, a hierarchy of keys is provided:

```
  OSD Key hierarchy:
      Master key
          |
      device key
          |
   Object Group key(s)
    /    |    \
 Object Group working key A
 Object Group working key B
```

~~When~~ Initially an OSD device ~~ships from the factory, it~~ 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. 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 ~~Requester~~ application client systems (that is, on a two-port network on the desk of an administrator).

Editor's note: How does the OSD know that is where it is?

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 ~~which~~ that modify this key.~~,~~ ~~which~~ 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 Object Groups on the same OSD ~~can~~ may be provided separate Object Group keys, and thus be totally denied access to one another's' Object Groups. Each Object Group also has two OBJECT GROUP WORKING KEYS, A and B~~, which~~ 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.

It is expected that most operations ~~which~~ that require the use of a key in the~~is~~ key hierarchy will use Object Group working keys, and that Policy/Storage managers will change the Object Group working keys regularly (for instance, 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 working key to use in future capability generation, the first of these working keys ~~can~~ may be changed without invalidating capabilities signed with the other~~, which~~. This allows capabilities generated shortly before a working key is changed to continue to work until they gracefully time out according to their expiration time.

C.6.4 Security Capabilities

~~Requester~~Application clients ~~(that is, not~~ except for Policy/Storage Managers~~)~~ ~~will~~ should never possess one of the keys in the OSD key hierarchy. Instead, they ~~will~~ should be issued individual capabilities by storage/policy managers. A capability describes the accesses that ~~which~~ ~~Requester~~ application clients are allowed to perform by proving that they have been given that capability. Capabilities are validated by means of a secure hash of the capability~~,~~ ~~which~~ that is available to the ~~Requester~~ application client, and a key from the key hierarchy, ~~which~~ that is not available to the ~~Requester~~ application client but is available to both the Policy/Storage manager and the targeted OSD. This computation is known as signing a hash, or a signature, because only the holder of the key ~~can~~ may generate the hash. Although an ~~Requester~~ application client ~~can~~does not generate this hash, it ~~can~~ may hold the hash and use it to prove to the OSD device that that the Policy/Storage Manager granted the associated capability. Because an ~~Requester~~ application client ~~can~~does not generate a correct hash and because the correct hash includes the capability fields as well as the secret key, an ~~Requester~~ application client that has been granted a specific capability ~~can~~does not change any values in the capability's fields and uses this new capability for interactions with the OSD device.

A central feature of this OSD capability system is that these signed hashes ~~can~~ 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 ~~Requester~~ application client ~~can~~ may prove 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.

~~This section is concerned only with defining capabilities; Section~~ C.2.5 discusses binding of capabilities to actions.

C.6.5 Security Capability format

Capabilities are 95 bytes long and contain the following fields:

Permissions	(32 bits)
Expiration time	(64 bits)
Device name	(64 bits)
Flavor select	(4 bits)
Key identifier	(4 bits)
Object Group	(8 bits)
Minimum security	(8 bits) (see C.2.5)
Flavor-specific	(288 bits)

As described above, the permissions word is a bit-wise OR of the operations permitted by this capability. The expiration time is a timestamp beyond which the OSD device should no longer honor this capability. The device name is the unique identifier of the OSD device. This ensures that a capability intended for use on one OSD ~~can~~ should not be used on another, even in the ~~(~~unlikely~~)~~ event of key duplication. The key identifier indicates what key in the OSD key hierarchy was used to generate the capability key-signature that ~~will~~ may be used to prove that the ~~Requester~~ application client has been granted this capability. The Object Group field indicates what Object Group this capability is valid for (some operations, such as Set Device Association, do not need to check for a match with this field). Like the keys in the OSD key hierarchy, the capability key-signature should never be transmitted over the network without encrypting it. The minimum security requirement mandates a minimum set of security requirements that any operation performed using this capability ~~shall~~should meet (see ~~section~~ C.2.6). Capability flavors (a type specifier) allow different definitions of the scope of a capability.

C.6.6 Permissions

Every capability includes an enumeration of what operations are to be permitted by it. This is in the form of a 32-bit word, where each bit represents an operation. If the bit is 1, the operation is permitted. If the bit is 0, this capability does not permit the operation.

Permission bits:

Bit 31 CREATE

Bit 30 OPEN / CLOSE

Bit 29 READ

Bit 28 WRITE

Bit 27 APPEND

Bit 26 FLUSH OBJECT

Bit 25 FLUSH OBJECT GROUP

Bit 24 REMOVE

Bit 23 CREATE OBJECT GROUP

Bit 22 REMOVE OBJECT GROUP

Bit 21 GET OBJECT ATTRIBUTES

Bit 20 SET OBJECT ATTRIBUTES

Bit 19 GET DEVICE ASSOCIATION

Bit 18 SET DEVICE ASSOCIATION

Bits 17..0 Reserved

C.6.7 Key identifier values

Values for the key identifier are:

0	Master key
1	Device key
2	Object Group key
3	Object Group working key A
4	Object Group working key B
5..7	~~r~~Reserved

The remaining eight values (i.e., bit 3 set in the key identifier field) are defined here for use in ~~Section ~~C.2.5. These values are not valid for the key identifier field of a capability.

8	capability key-signature with signature basis key = master key
9	capability key-signature with signature basis key = device key
10	capability key-signature with signature basis key = object group key
11	capability key-signature with signature basis key = object group working key A
12	capability key-signature with signature basis key = object group working key B
13..15	R~~r~~eserved

C.6.8 Flavors

C.6.8.1 Overview

A capability ~~shall ~~also indicates on what object(s) the operations indicated by its permissions are authorized. There is more than one way in ~~which ~~that a capability may do so. These ways are known as flavors. This proposal offers three object defining schemes: no object (OSD only), one object, and a set of objects determined by matching against attribute fields that are uninterpreted by the OSD device.

The flavor of a capability is determined by the flavor-select field. Valid values for this field are:

0	Null
1	Single-range
2	Match
3..15	Reserved

Each individual flavor defines the meaning of the bits in the flavor-specific field of the capability. All bits of the flavor-specific field, including those that are not used by the individual flavor, are included in the computation of the capability key-signature.

C.6.8.2 Null flavor

The flavor-specific field is entirely unused. This flavor is primarily intended for operations (such as CREATE OBJECT GROUP, get/set device association, etc) ~~which~~ that do not refer to a particular object.

Definition: The operations enumerated in the permissions word are permitted on the named device, restricted to the named Object Group (when applicable), before the specified expiration time.

C.6.8.3 Single-range flavor

The single-range flavor is designed to allow access to a single object, or a range of bytes within a single object.

The flavor-specific field of the single-range flavor contains:

object identifier	(64 bits)
offset	(64 bits)
length	(64 bits)
access control state	(16 bits)
object creation time	(64 bits)

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

~~9~~ All requirements of the Null flavor are met.

~~10~~The access control state in the capability matches the access control state in the object's attributes. This enables simple revocation of a capability by a Policy/Storage Manager.

~~11~~If the length is nonzero, operations upon the data contents of the object, such as READ, WRITE, and APPEND, may only operate upon bytes in the range [offset, offset+length-1]. In the case of append, the object may be extended from its current length but may not be extended to a length exceeding offset+length-1.

~~12~~If the length is 0, operations may not operate on bytes in the range [0, offset-1]. This makes no restriction if offset is also 0.

~~13~~13 If the object creation time of the capability is non-zero, then the creation time attribute of the object shall be equal to the object creation time field of the capability. With this restriction, this definition of a capability is not adversely impacted by changing the designator of an object's ID from the OSD device, as it is in this document, to the ~~Requester~~ application client, as some have advocated.

Editor's note: What does the last sentence mean?

C.6.8.4 Match flavor

The match flavor is defined~~signed~~ to allow access to multiple objects satisfying predetermined properties. This may dramatically reduce the frequency ~~that~~ of ~~Requester~~ application clients ~~shall~~ obtaining new capabilities from Policy/Storage Managers. Because this flavor allows the capability creator to specify four separate values that ~~shall~~ 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

~~14~~all requirements of the null flavor are met, and

~~15~~for each N {0,1,2,3}: the bitwise AND of maskN and the 4 bytes starting at byte offsetN of the FS-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 FS-specific field of each object.

C.6.9 Revisiting byte 5 of the long CDB, the encryption identifier

~~Some assert that t~~The size of the encryption identifier, byte 5 of the long CDB, may be~~is~~ too small. For example, since the Object Group ID is 8 bits in size, it is not possible for each Object Group to have a distinct key and name this key in the encryption identifier without consuming all the information space of this identifier, that~~which~~ should also specify at least the absence of encryption. It is desirable to have a separate key for each Object Group in ~~systems in which~~ when Object Group~~'~~s are assigned to non-cooperating application managers ~~which~~ that each believe they have a dedicated OSD at their disposal.

~~In the sub-sections that follow b~~Byte 5 of the long CDB may be ~~is~~ ignored and its functions ~~are redundantly specified~~overridden by additional requirements. One of the 256 values of byte 5 of the long CDB ~~could~~ may indicate that the following scheme is to be employed, for example. Moreover, for completeness, the security description of C.2 does not how CDBs (and action arguments) ~~are~~ to be increased to accommodate the security arguments. This ~~shall~~ should be done before security mechanisms requiring more than 8 bits of attribute values ~~can~~ may be employed by ~~OBS~~OSA systems.

C.6.10 Securing operations

When an ~~Requester~~ application client sends a CDB to an OSD device, it sets bits in a security-type attribute in the CDB describing the security provided ~~on~~ for the request. This security-type also serves to specify the minimum security requirement for the response from the OSD device. The security-type attribute ~~shall~~ should be transmitted in clear text, since it specifies the keys and algorithms that apply to the rest of the command.

The security-type attribute contains the following fields:

Key identifier	(4 bits)
Object Group	(8 bits)
Security level	(16 bits)

The key identifier field is identical to the key identifier specified in C.2.3.3 and allows all 16 values. The key specified by the key identifier field ~~will~~ may be referred to as the Operation key. ~~Note that t~~The operation key (including the capability key-signature, if that is the key being used) ~~itself~~ is not transmitted in most commands. The OSD device either computes it from the information included in the capability (for a capability key-signature) or retrieves it from local key storage (for one of the other keys).

The object group field indicates to which object group an object-group-specific key applies. This is significant only when the key identifier indicates a object group key, object group working key, or capability key-signature which uses the object group key or object group working key.

Requests to the OSD device may be authorized either directly by one of the keys from the OSD device's key hierarchy, ~~which we will refer~~refereed to as a DIRECT KEY, or by specifying a capability. Since ~~Requester~~ application clients will typically not hold any key in the OSD key hierarchy, it is likely that only Policy/Storage will never issue requests that use a direct key. However, this facility is provided to allow a manager to issue requests directly to the OSD device without the overhead associated with constructing a capability. A request specified with a direct key should be treated by the OSD device as equivalent to a null-flavored capability with the permission bits set to all ones.

The security level field of the security-type indicates which bytes transferred are signed and/or encrypted. If a transferred byte is signed, then its value is included in a signed hash computation and the resulting hash value (signature) is also sent in order for the receiver to verify it. The signed hash size is 128 bits and it is sent after the fields it verifies.

The key used for signatures or encryption is the operation key, as defined above. The only exception to this is an encrypted capability, when the specified operation key is a capability key-signature. In this case, an OSD device needs to be able to read the capability in order to compute the capability key-signature.~~, so t~~The capability cannot be encrypted with the yet-to-be-determined key. Instead, the BASIS KEY for the capability key-signature is used to encrypt and decrypt the capability. That is, if the operation key is "capability key-signature from object group working key A", then the capability is encrypted with object group working key A. Once the capability has been decrypted, the OSD device ~~can~~ may compute the corresponding capability key-signature and use that as the operation key for all subsequent encryption and signature operations. Capabilities do not need to be signed separately from the rest of the command arguments because signatures do not obscure the capability’s values.

The bits in the security level field are defined as:

9..15	Reserved
8	Capability is encrypted
7	Arguments are signed

6 Arguments are encrypted
5 Data-in are signed
4 Data-in are encrypted
3 Data-out shall be signed
2 Data-out shall be encrypted
1 Result shall be signed
0 Result shall be encrypted

When arguments are signed, the data-in phase ~~shall~~ should begin with a signature of all the bits (including the first 8 bytes) in the CDB. When data-in is signed, the data-in phase ends with a signature of all the bytes in the CDB and the data-in phase, excluding the signature of the CDB ~~(~~if present~~)~~.

If arguments are encrypted, all bytes in the CDB following first 8 clear text bytes and excluding the security-type attribute field are encrypted. If data-in is encrypted, all bytes transferred as part of the data-in phase.

The data-out phase shall begin with a byte of security information, which is defined as:

Bits 7..4 Reserved
Bit 3 Data-out is signed
Bit 2 Data-out is encrypted
Bit 1 Result is signed
Bit 0 Result is encrypted

If the data-out buffer is signed, the data-out ~~phase~~ buffer transfer ends with a signature of all the bytes in the data-out phase, including the security information byte. If the result is signed, the data-out buffer signature is followed by a signature of the result block. If the data-out buffer is encrypted, all bytes transferred during the data-out~~ phase~~buffer transfer, including the data-out buffer and result signatures, are encrypted.

Note that when sending large amounts of data in the data-in buffer and data-out buffer ~~phases~~transfers, it may be necessary to add intermediate signatures to validate portions of the data to avoid buffering problems. One approach is to mandate a signature every N bytes in the data stream. Another approach is to mandate a signature every time the data stream for a Read, WRITE, or APPEND crosses a logical boundary; for example, the end of a scatter/gather component, or the offset within the object associated with the current byte in the stream crosses an M byte boundary. ~~The union of these signature requirements is what was used in CMU's prototype.~~

C.6.11 Minimum Security Requirements

A MINIMUM SECURITY REQUIREMENT is an enumeration of what security ~~shall~~ should be present for a request to be accepted and potentially succeed. The minimum security requirement field defines the same values as the security level field (see C.6.10~~ C.2.5~~).

Each object group has a minimum security requirement as an attribute. Actions within that object group are only permitted if the meet at least the minimum security requirement.

Each capability also has a minimum security requirement. This allows policy/storage managers to ~~force~~ instruct ~~Requester~~ application clients to use protection when performing certain operations.

C.6.12 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. ~~We recommend adding a new operation,~~ SET KEY should be added~~,~~ for this purpose~~, though it is conceivable that the SET ATTRIBUTE operation could be overloaded to provide this functionality~~.

The SET KEY operation has a minimum security requirement of all 1s (i.e., encryption and authentication are mandatory). ~~It shall~~ 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. ~~Some discussion has taken place as to whether alteration of the master key should ever be allowed, or whether it should be immutable. In this argument, l~~Loss of the master key should be considered a catastrophic failure and the device destroyed, security erased, or ~~at least~~ reformatted~~, obliterating all data~~ 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 ~~all~~ routine operations. This avoids unnecessary exposure of high-level keys.

Annex D Motivation for the NSIC OSD (Informative)

D.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 use~~s~~d 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~~ at end of Clause~~ in the bibliography.

The NSIC/NASD Working Group ~~has ~~chose~~n~~ 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 ~~ document~~standard.

D.2 Potential OSD Products

The NSIC/NASD Working Group ~~is ~~defined~~ing~~ an architecture for object-based storage devices. The definition deliberately encompassed~~s~~ the concept of *virtual* OSD devices exported by storage aggregators (e.g., RAID controllers) as well as actual storage devices (e.g., disc~~k~~ drives, ~~and ~~tape drives, and solid state disc~~k~~s). ~~It is entirely possible that t~~The first OSD "storage devices" introduced to the market ~~will in fact ~~may be virtual storage devices instantiated by controllers managing conventional disc~~k~~s. While sacrificing the fine scaling granularity that would result from physical OSD devices, the controller approach ~~would ~~may have the advantage of deferring the necessity to implement host- or Policy/Storage Manager-assisted aggregation in order to gain OSD benefits.

D.3 Benefits of OSD

The NSIC/NASD Working Group ~~is ~~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:

~~It might seem that there is nothing in the problems of clustering or Storage Area Networks that dictates the use of Object based storagethe Object Storage Architecture to solve them. In fact object based storage does so much to improve the cluster architecture that NSIC feels it is essential to realize the full benefits of the clustered system architecture. Though this is far from doing the subject justice, here are several reasons for this position.~~

~~1.~~Higher quality storage management operations with less host effort;

~~Objects really make the self-management of storage possible. Without the storage device having sufficient knowledge of the resident data, it cannot assume the responsibility for managing space. Storage devices could not participate in any attribute management without the knowledge of what constitutes a meaningful subset of its space or when it is appropriate to take action. More effective management will result from the storage devices able to participate intelligently.~~

~~If management policies can be communicated to the storage device so that it can act independently to execute them, the result will be not only~~ less human intervention ~~required but also~~

tighter and more timely control;~~.~~

~~Consider the case of weekly backup. Systems are usually backed up during an idle period on weekends, so that the system availability is not interrupted during the business week. This also produces a backup that is guaranteed to be consistent. This window has been gradually shrinking at the same time system capacities have been exploding. Trying to find time to interrupt a system long enough to backup possibly terabytes of data has become an almost insoluble problem.~~

~~By taking action on an object based on attributes assigned to it, The OSD device could inform a backup function whenever an object has reached the correct state for its backup to be taken. The backup of all files could be spread over a longer period - during which others are still being updated -without affecting data integrity.~~

~~Often, it is not quite so simple. It may be that several objects constitute an interdependent set that must be backed up together and only when all have reached a consistent state. Consider a database consisting of 6 files, none of which can be backed up until either all have been closed or until one designated as the object on which all of the others are dependent has been closed. A Policy/Storage Manager may be needed to manage this kind of relationship between Objects. In this case, when one of the objects has been updated, an attribute could cause a signal to a this manager that would result in an attribute set in a different object. If each of the other five objects in this set were to do this, the OSD device could act on all the required attributes being set in the sixth to initiate a backup process.~~

~~Other attributes that could invoke action by the OSD device include encryption, compression, versioning, parity redundancy and HSM style migration. In each of these, the storage device would only have to be informed of the policy with respect to a specific object or set of objects. It could then perform the function itself or inform an agent designated to provide that service.~~

~~Compression and encryption could be done right on the OSD device, so that only the fact that one is required for an object need be communicated to the storage device. For a management function that must go off the drive, such as HSM, not only the policy is needed but also the identification of an agent to perform the function.~~

~~2.The~~ controlled sharing of data; ~~can be controlled more efficiently when the storage device knows what constitutes an entity.~~

~~If two systems were to share a BBSD, all the meta-data~~attributes ~~activity would have to be serialized for concurrent access. In an OSD device much of the meta-data~~attributes ~~activity is opaque to the systems, which need only concern themselves with access conflicts to user data or file directories. Also space management being done by the storage device eliminates any contention or confusion that could arise from two systems trying to allocate space on the same storage device at the same time.~~

~~3.~~easier ~~H~~heterogeneous computing; ~~should be made much easier by an object abstraction.~~

~~There is essentially no commonality among OSs meta-data~~attributes ~~structures. OSD should make it possible to have common foundation on which any OS can overlay its file system. An OSD device enforces a host-independent concept of storage objects. It is therefore possible for any host with a file system supporting OSD storage to share storage devices with other interconnected hosts, even if their file structures are incompatible~~

~~4.~~1. ~~M~~minimize~~s~~ data access synchronization requirements for clustered servers;~~.~~

~~Since each read or WRITE to an OSD device is within the context of an object, the OSD device itself enforces the isolation of servers' data accesses in "shared nothing" clusters. In shared data clusters, OSD can minimize inter-server lock traffic by allowing the use of optimistic locking protocols.~~

~~5.While there are a lot of issues revolving around~~ performance benefits in a clustered system architecture~~, some obvious performance benefits come with the Objects.~~ :

~~a.~~The ~~meta-data~~attributes never leaves the storage device, eliminating a certain amount of I/O.

~~b.~~The storage device may knows which objects are closed or open, and is able to use that information to more effectively manage its cache.

~~c.~~Prefetching ~~can~~ may be much more effective as the storage device knows the layout of the object being read. The storage device ~~can~~ may more effectively determine sequential access patterns ~~– or could be told of sequentiality~~.

~~d.~~The cache in the storage device ~~can~~ may hold ~~meta-data~~attributes once for multiple systems accessing it.

~~e.~~The storage device ~~can~~ may participate in quality of service decisions, such as where to locate data most appropriately. ~~It can only do this if it has responsibility for allocating storage. By comparison, few OSs can allocate data by zone on a disc drive.~~

~~f.~~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

E

~~References~~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