ENDL
TEXAS

Date: 14 June 2008
To: SNIA OSD TWG and T10 Technical Committee
From: Ralph O. Weber
Subject: OSD-2 CDB Continuations Definition and Usage

Introduction

The recent SNIA OSD TWG face-to-face meeting agreed to add a scatter/gather capability to OSD-2 Read and Write commands. However, the attributes-based method for describing the scatter/gather list (see T10/08-091r1) was found to present data ordering problems in the Data-Out Buffer.

A new CDB Continuation method was agreed upon to provide space for the scatter/gather list. The new method more nearly matches the concept of a SCSI Parameter List, but based on the existing OSD security mechanisms changes that more specifically match the OSD Data-Out Buffer definition are needed.

This proposal describes:

- The appropriate Data-Out Buffer format changes (see change 1);
- How the existing CDB is continued into the modified Data-Out Buffer (see change 2);
- How sets of capabilities can be placed in the CDB and CDB continuation segment of the Data-Out Buffer (see change 3);
- Increasing all integrity check value field sizes so that HMAC-SHA-256 hash outputs can be used (see change 4)

Based on the above general enhancements of the OSD CDB, the following new command features are defined:

- A scatter/gather list addition to the CREATE AND WRITE command, READ command, and WRITE command (see change 5); and
- A new COPY USER OBJECTS command (see change 7) and supporting object duplication model (see change 6).

These changes to the suite of OSD commands provide useful new OSD functions. They also demonstrate how the new CDB continuation feature can be used by other new and existing OSD commands.

Revision History

r0 Initial revision

r1 Incorporate changes requested by the March CAP working group.

08-105r1 was approved for incorporation in OSD-2 r04 by the March T10 Plenary.
Subsequent to that new errors were discovered by the SNIA OSD TWG.

<u>08-158 was begun to address those errors.</u>

r0
- The CDB continuation length was moved out of the Data-Out Buffer into the CDB.
- Multiple individual capabilities removed from CDB continuation and replaced with a single set of one or more capabilities.
- Credential format updated to support coordinated delivery of a main (CDB) capability and set of one or more additional capabilities.

18484 Preston Road, Suite 102 PMB 178, Dallas, TX 75252 214-912-1373 972-596-2775 Fx ENDL_TX@computer.org

r1 Several relatively minor changes were made:
- A conflicting ALLDATA definition was removed from change 1 because the definition in change 3 was more up to date.
- The ALLDATA definitions were updated to keep the CDB continuation segment out of the ICV computation because it has already been checked separately.
- The integrity check value field size changes that were incomplete in r0 were completed, mostly in change 4.
- The entire proposal was reviewed and several clarifying changes were made.

r2 Incorporated changes requested by the SNIA OSD TWG. Also corrected the length of capabilities to be 104 bytes in all occurrences, and eight-byte aligned the first byte of all capabilities.

Differences between r1 and r2 are identified by change bars.

Unless otherwise indicated additions are shown in blue, deletions in ~~red strikethrough~~, and comments in green.

Each change is introduced by a brief description in which the markups described above are not used.

Change 1 – Data-Out Buffer format and related changes

Description

This change splits the Data-Out Buffer command data or parameter data segment into specific subsegments for each function. What SCSI would normally call the parameter data segment is called the CDB continuation segment. In some commands (e.g., WRITE) both the CDB continuation segment and the command data segment may be present in one Data-Out Buffer.

Proposed Changes in OSD-2 r03

4.14.2 OSD meta data

A single command may include the following types of data:

~~a) Traditional command data or parameter data;~~
a) Traditional parameter data, that is placed in a CDB continuation segment;
b) Traditional command data;
c) OSD object meta data; and/or
d) Integrity check values computed over all the other types of data.

The presence of generalized object meta data differentiates communications in the OSD model from those used by traditional block structured devices (i.e., SBC devices).

NOTE 4 This standard provides for several segments in the Data-in and Data-out Buffers because the output ~~meta~~ data is typically too large to fit in the CDB, the input meta data is too large to fit in the single status byte returned by SCSI devices, and the ALLDATA security method (see 4.12.4.5) provides for the computation of integrity check values for all data bytes exchanged between the application client and device server.

OSD meta data and integrity check values share the Data-In Buffer and Data-Out Buffer with the traditional command or parameter data as shown in table 34.

Table 34 — OSD Data-In Buffer and Data-Out Buffer model

Bit / Byte	7	6	5	4	3	2	1	0
0				Command data and/or parameter data segment, if any				
i-1								
i				Unused bytes, if any				
k-1								
k				Meta data segments, if any				
p-1								
p				Unused bytes, if any				
m-1								
m				Integrity check value segment, if any				
n								

The Data-In Buffer format is described in 4.14.3. The Data-Out Buffer format is described in 4.14.4.

Offset values (see 4.14.5) for the various segments ~~each segment except the first~~ are provided in CDB fields. The segments of the Data-In Buffer and Data-Out Buffer should not overlap. If they do, the results are unpredictable.

4.14.3 OSD Data-In Buffer format

{{Although not strictly necessary, the changes proposed for 4.14.3 are intended to make its contents consistent with those of 4.14.4. In 4.14.4, the proposed changes are needed to clarify the use of the newly defined buffer segment and the existing buffer segment.}}

The Data-In Buffer has the format shown in table 35.

Table 35 — OSD Data-In Buffer format

Bit / Byte	7	6	5	4	3	2	1	0
0				Command data or parameter data segment, if any				
i-1								
i				Unused bytes, if any				
k-1								
k				Retrieved attributes segment, if any				
p-1								
p				Unused bytes, if any				
m-1								
m				Data-In Buffer integrity check value segment, if any (see 4.12.4.5)				
n								

The command data or parameter data segment contains data transferred from an object to the application client (e.g., data read by a READ command (see 6.23)) or data returned to the application client by the device server in response to a request made by the command (e.g., the matches list parameter data returned by a QUERY command (see 6.21)).

The retrieved attributes segment contains attribute values retrieved based on requests specified by the CDB (see 5.2.4).

The Data-In Buffer integrity check value segment contains security parameters related to the ALLDATA security method (see 4.12.4.5).

The CDB offset fields that assist in locating the Data-In Buffer segments are shown in table 36.

Table 36 — Summary of OSD Data-In Buffer offsets

CDB Data-In Buffer offset field	Reference	Buffer segment
none		Command data or parameter data
RETRIEVED ATTRIBUTES OFFSET	5.2.4	Retrieved attributes data
DATA-IN INTEGRITY CHECK VALUE OFFSET	5.2.8	Data-In Buffer integrity check value

If the device server sends data to the unused Data-In Buffer bytes in the initiator device, then the device server shall send bytes containing zero.

4.14.4 OSD Data-Out Buffer format

The Data-Out Buffer has the format shown in table 37.

Table 37 — OSD Data-Out Buffer format

Bit / Byte	7	6	5	4	3	2	1	0
0				~~Command data or parameter data~~ CDB continuation segment (see 5.x), if any				
h-1								
h				Command data segment, if any				
i-1								
i				Unused bytes, if any				
k-1								
k				Set attributes segment, if any				
x-1								
x				Unused bytes, if any				
y-1								
y				Get attributes segment, if any				
z-1								
z				Unused bytes, if any				
m-1								
m				Data-Out Buffer integrity check value segment, if any (see 4.12.4.5)				
n								

The CDB continuation segment contains fields that elaborate on the command to be processed (see 5.x).

The command data segment contains data to be transferred to an object from the application client (e.g., data to be written by a WRITE command (see 6.32)).

The set attributes segment contains attribute values to be set based on requests specified by the CDB (see 5.2.4).

The get attributes segment contains a list of attribute values to be retrieved based on requests specified by the CDB (see 5.2.4).

The Data-Out Buffer integrity check value segment contains security parameters related to the ALLDATA security method (see 4.12.4.5).

The CDB offset fields that assist in locating the Data-Out Buffer segments are shown in table 38.

Table 38 — Summary of OSD Data-Out Buffer offsets

CDB Data-Out Buffer offset field	Reference	Buffer segment
~~none~~		~~Command data or parameter data~~
none		CDB continuation
CDB CONTINUATION LENGTH [a]	5.2.1	Command data
SET ATTRIBUTES LIST OFFSET	5.2.4	Set attributes
SET ATTRIBUTES OFFSET	5.2.4	Set attributes
GET ATTRIBUTES LIST OFFSET	5.2.4	Get attributes
DATA-OUT INTEGRITY CHECK VALUE OFFSET	5.2.8	Data-Out Buffer integrity check value
[a] This is a count of the bytes in the preceding segment, not an offset.		

The device server shall ignore the contents of unused bytes in the Data-Out Buffer.

…

5.x CDB continuation segment format

{{All of 5.x is new. The use of change markups is suspended for the remainder of 5.x.}}

If the CDB CONTINUATION LENGTH field (see 5.2.1) is not set to zero, the first bytes in the Data-Out Buffer (see 4.14.4) have the format shown in table x1.

Table x1 — CDB continuation segment format

Bit / Byte	7	6	5	4	3	2	1	0
0	CDB CONTINUATION FORMAT							
1	Reserved							
2	(MSB)			CONTINUED SERVICE ACTION				
3								(LSB)
4	Reserved							
7								
8	(MSB)			CONTINUATION INTEGRITY CHECK VALUE				
39								(LSB)
	CDB continuation descriptors							
40	CDB continuation descriptor [first] (see 5.y.1)							
	⋮							
	CDB continuation descriptor [last] (see 5.y.1)							
	Pad bytes (for application client alignment)							
n								

The CDB CONTINUATION FORMAT field (see table x2) specifies the format of this CDB continuation segment.

Table x2 — CDB CONTINUATION FORMAT field

Value	Description
00h	Reserved
01h	The format defined by this standard
02h to FFh	Reserved

The CONTINUED SERVICE ACTION field specifies the service action for the command to which the CDB continuation is being applied. If the contents of the CONTINUED SERVICE ACTION field do not match the contents of the SERVICE ACTION field in the CDB (see 5.1), then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The CONTINUATION INTEGRITY CHECK VALUE field contains an integrity check value (see 4.12.8)(see 4.12.8) for the CDB continuation segment sent by the application client. The CONTINUATION INTEGRITY CHECK VALUE field is used for all security methods (see 4.12.4) except NOSEC.

Each CDB continuation descriptor (see 5.y.1) contains one set of CDB continuation information. Unless otherwise stated, the order in which the CDB continuation descriptors appear in the CDB continuation segment has no significance.

The CDB continuation segment may be padded to meet alignment requirements determined by the application client. Depending on the needed alignment, zero or more bytes containing zeros may be added at the end of the CDB continuation segment.

5.y CDB continuation descriptors

{{All of 5.y is new. The use of change markups is suspended for the remainder of 5.y.}}

5.y.1 Overview

Each CDB continuation descriptor (see table x3) contains one set of CDB continuation information formatted as a header with a format that is common to all CDB continuation descriptors followed by data that is specific to each CDB continuation descriptor type.

Table x3 — CDB continuation descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
	CDB continuation descriptor header							
0	(MSB)			CDB CONTINUATION DESCRIPTOR TYPE				
1								(LSB)
2	Reserved							
3	Reserved					PAD LENGTH (p-n)		
4	(MSB)			CONTINUATION DESCRIPTOR LENGTH (n-7)				
7								(LSB)
	CDB continuation descriptor type specific data							
8	CDB continuation descriptor type specific data							
n								
	CDB continuation descriptor alignment bytes							
n+1	Pad bytes (for eight-byte alignment)							
p								

The CDB CONTINUATION DESCRIPTOR TYPE field (see table x4) specifies the format of the CDB continuation descriptor type specific data.

Table x4 — CDB CONTINUATION DESCRIPTOR TYPE field

Value	Description	Reference
0000h	No more continuation descriptors [a]	
{{Other rows of this table are filled in by other changes in this proposal.}} {{The most complete body for this table can be found in change 7 table x4.}}		
all other values	Reserved	

[a] Since the CDB continuation segment pad bytes, if any, are set to zero (see 5.x), encountering a CDB continuation descriptor type of zero shall be processed in the same way as reaching the last byte of the CDB continuation segment.

The PAD LENGTH field specifies the number of bytes containing zeros that follow the CDB continuation descriptor type specific data.

The CONTINUATION DESCRIPTOR LENGTH field contains the number of bytes of CDB continuation descriptor type specific data that follow in this descriptor.

If the sum of the pad length and the continuation descriptor length is not a multiple of eight, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The format of the CDB continuation descriptor type specific data depends on the contents of the CDB CONTINUATION DESCRIPTOR TYPE field (see table x4).

…

7.1.2.8 Root Information attributes page

The Root Information attributes page (R+1h) shall contain the attributes listed in table 127.

Table 127 — Root Information attributes page contents

Attribute Number	Length (bytes)	Attribute	Application Client Settable	OSD Logical Unit Provided
…	…	…	…	…
9h	variable	OSD name	Yes	No
Ah	8	Maximum CDB continuation length	No	Yes
~~Ah~~ Bh to 7Fh		Reserved	No	
80h	8	Total capacity	No	Yes
…	…	…	…	…

…

The maximum CDB continuation length attribute (number Ah) shall contain the largest value allowed in the CDB CONTINUATION LENGTH field (see 5.x).

…

{{The maximum CDB continuation length attribute must be added to Annex B too.}}

7.1.2.11 User Object Information attributes page

…

If the OSD logical unit does not support the reserved data space attribute, the actual data space attribute (D1h) shall be undefined (see 3.1.50). If the reserved data space attribute is supported, the actual data space attribute shall be defined (see 3.1.15) and shall contain the number of bytes used by the user object to store data transferred in the command data ~~or parameter data~~ segment of the Data-Out Buffer (see 4.14.4) by APPEND commands (see 6.2), CLEAR commands (see 6.3), CREATE AND WRITE commands (see 6.5), and/or WRITE commands (see 6.32) to the user object.

…

Change 2 – CDB Continuation into the Data-Out Buffer

Description

This change adds a field to the basic OSD CDB format to specify the size (in bytes) of a CDB continuation segment in the Data-Out Buffer. Unlike other buffer segments, the CDB continuation segment and the command data segment which follows it do not use offset fields to locate them. If the CDB continuation segment is present, it is immediately followed by the command data segment, and any needed alignment is achieved by padding the CDB connotation segment.

Proposed Changes in OSD-2 r03

5.2 Fields commonly used in OSD commands

5.2.1 Overview

OSD commands employ the basic CDB structure shown in 5.1. Within the basic CDB structure, the OSD service action specific fields are organized so that the same field is in the same location in all OSD CDBs (see table 49). OSD service action specific fields that are unique to a small number of CDBs are not shown in this subclause.

Table 49 — OSD service action specific fields

Bit Byte	7	6	5	4	3	2	1	0
10	Reserved			DPO [a]	FUA [a]	ISOLATION (see 5.2.5)		
11	Reserved		GET/SET CDBFMT [b]		Command specific options			
12	TIMESTAMPS CONTROL (see 5.2.10)							
13 15				Reserved				
16 23	(MSB)			PARTITION_ID (see 5.2.7)				(LSB)
24 31	(MSB)			USER_OBJECT_ID (see 5.2.11)				(LSB)
32 39	(MSB)			LENGTH (see 5.2.6) or ALLOCATION LENGTH (see 5.2.2)				(LSB)
40 47	(MSB)			STARTING BYTE ADDRESS (see 5.2.9)				(LSB)
48 51	(MSB)			CDB CONTINUATION LENGTH (see 5.2.x)				(LSB)
48 51				~~Reserved~~				
52 …				{{No other changes in the body of table 49.}}				

[a] See 5.2.3.
[b] See 5.2.4.

{{Locating the CDB CONTINUATION LENGTH field in bytes 48 to 52 produces conflicts with the LIST and LIST COLLECTION commands LIST IDENTIFIER field; the READ MAP command REQUESTED MAP TYPE field; the SET KEY command SEED field; and the QUERY command QUERY LIST LENGTH field. With the exception of the QUERY command the conflicts are being ignored, and this means that the LIST, LIST COLLECTION, READ MAP, and SET KEY commands will never be able to have CDB continuations. The conflict in the QUERY command is addressed by 08-182.}}

…

{{Insert the following new subclause in the proper alphabetical order.}}

5.2.x CDB continuation length

The CDB CONTINUATION LENGTH field specifies the number of bytes in the CDB continuation segment of the Data-Out Buffer (see 4.14.4) using the format described in 5.x.

The command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB, if the CDB CONTINUATION LENGTH field contains a non-zero value that is:

a) Not a multiple of eight;
b) Less than 48 (i.e., 30h); or
c) Greater than the value in the maximum CDB continuation length attribute in the Root Information attributes page (see 7.1.2.8).

…

6 Commands for OSD type devices

{{For all command definitions except LIST, LIST COLLECTION, QUERY, READ MAP, SET KEY, CREATE AND WRITE, READ, and WRITE, the CDB format definition table and text that follows it must be modified as shown. The changes needed in the CREATE AND WRITE command, READ command, and WRITE command are shown in change 5. The changes needed in the QUERY command are in 08-182. The LIST, LIST COLLECTION, READ MAP, and SET KEY commands already use the bytes where the CDB CONTINUATION LENGTH field is being placed and thus cannot have CDB continuations without major rewrites.}}

Table nn — generic OSD command definition table

Bit / Byte	7	6	5	4	3	2	1	0
…	…							
48	Reserved							
51								
48	(MSB)							
51	CDB CONTINUATION LENGTH (see 5.2.x)							(LSB)
52	{{No other changes in the body of this table.}}							
…								

…

The contents of the CDB CONTINUATION LENGTH field are defined in 5.2.x. If the CDB CONTINUATION LENGTH field contains a value other than zero, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The get and set attributes parameters are defined in 5.2.4. …

Change 3 – Placing additional capabilities in the CDB continuation segment

Description

The M_OBJECT permission bit allows the manipulation of the attributes in multiple user objects, but not manipulation of the data. This could be taken to mean that manipulation of data (i.e., the information for which a READ or WRITE permission bit is required) should be granted only if a full capability is provided for the object (user object, collection, or partition) that is being accessed. If this view it taken, then sets that contain multiple capabilities need to be allowed, with one of these capabilities being placed in the CDB and the reset in the CDB continuation segment.

The changes below define:

- A credential format that allows sets of capabilities to be returned, along with two capability keys (one for the individual capability destined for the CDB, and a second for the entire set of capabilities), and
- A CDB continuation descriptor that contains the set of one or more capabilities returned in the modified credential format.

The entire capability definition subclause is modified so that all the capabilities of a command (the one in the CDB and any found in the set provided in the CDB continuation descriptor) are checked, instead of the current single capability check.

Warning: The subclauses changed by change 3 are shown out of numeric order to facilitate reviewing the changes.

Proposed Changes in OSD-2 r03

{{Place all glossary entries in proper alphabetical order.}}

3.1.4 capability: The fields in a CDB or CDB continuation segment (see 5.x) that specify what command functions (see 3.1.10) the command may request (e.g., what OSD object (see 3.1.29) may be accessed). The contents of capabilities may be managed for application clients by a policy/storage manager (see 3.1.34) and secured in credentials (see 3.1.11) by a security manager (see 3.1.41). See 4.11.2.

3.1.5 capability key: ~~The value in the CREDENTIAL INTEGRITY CHECK VALUE field (see 3.1.12)~~ An integrity check value computed for one or more capabilities and sent to an application client in a credential (see 3.1.11) that is used by ~~an~~ the application client to compute integrity check values for ~~a single~~ an OSD command. See 4.12.5.2.

…

3.1.11 credential: A data structure that is prepared by the security manager (see 3.1.41) and protected by an integrity check value (see 3.1.19) that is sent to an application client in order to grant defined access to an OSD logical unit for specific command functions (see 3.1.10) performed on specific OSD objects. The credential includes a capability (see 3.1.4) that is prepared by the policy/storage manager (see 3.1.34) that the application client copies to each CDB that requests the specified command functions. ~~See 4.12.5.1.~~ See 4.12.5.

{{There no longer is a single credential integrity check value. All references changed to describe the specific ICV that applies in context, or to a generic wording such as 'an integrity check value in a credential".}}

~~**3.1.12 credential integrity check value:** The integrity check value (see 3.1.19) protecting a credential (see 3.1.11). When the application client uses the credential integrity check value to compute integrity check values for a single OSD command, the value is called a capability key (see 3.1.5). See 4.12.5.1.~~

…

3.1.a extension capability: A capability (see 3.1.4) that is not the first capability returned in a credential (see 3.1.11) and is placed in the CDB continuation segment (see 5.x). A credential binds one or more extension capabilities to one solo capability (see 3.1.b).

3.1.b solo capability: A capability (see 3.1.4) that is the first capability returned in a credential (see 3.1.11) and is placed in the CDB (see 5.2.1). A credential may bind a solo capability to one or more extension capabilities (see 3.1.a).

…

4.12.5 Credentials

4.12.5.1 Credential format

A credential (see table 31) is transferred from the security manager to an application client over a communications mechanism that meets the requirements specified in 4.12.2. {{The capability sizes in OSD-2 r3 table 31 are 80 bytes, but they should be 104 bytes.}}

Table 31 — Credential format

Bit / Byte	7	6	5	4	3	2	1	0
0				~~Capability~~				
103				Solo capability (see 4.11.2.2)				
104				OSD SYSTEM ID				
123								
124	(MSB)			SOLO CREDENTIAL INTEGRITY CHECK VALUE				
155								(LSB)
156	(MSB)			EXTENSION CAPABILITIES LENGTH (k-159)				
159								(LSB)
160				Extension capability (see 4.11.2.2) [first]				
263								
				⋮				
k-103				Extension capability (see 4.11.2.2) [last]				
k								
k+1	(MSB)			EXTENDED CREDENTIAL INTEGRITY CHECK VALUE				
k+32								(LSB)

{{Both ICVs are 32-byte ICVs as described in change 4.}}

The solo capability is a capability ~~is described in 4.11.2.2~~ (see 4.11.2.2) to be copied to a CDB (see 5.2.1).

The OSD SYSTEM ID field specifies the value in the OSD system ID attribute in the Root Information attributes page (see 7.1.2.8) of the OSD logical unit to which the credential applies.

The SOLO CREDENTIAL INTEGRITY CHECK VALUE field contains an integrity check value (see 4.12.8) that is computed using the algorithm, inputs, and secret key specified in ~~4.12.6.3~~ 4.12.6.3.1.

The EXTENSION CAPABILITIES LENGTH field specifies the number of bytes that follow in zero or more extension capabilities.

Each extension capability is a capability (see 4.11.2.2). All the extension capabilities in a credential are copied to an extension capability CDB continuation descriptor (see 5.y.z).

The EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field contains an integrity check value (see 4.12.8) that is computed using the algorithm, inputs, and secret key specified in 4.12.6.3.2. If the EXTENSION CAPABILITIES LENGTH field is set to zero, then the EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field should be set to zero.

4.12.5.2 Capability ~~key~~ keys

All security methods except the NOSEC security method require the computation of one or more integrity check values using ~~a capability key~~ one or both of the integrity check values in the credential (see 4.12.5.1) as ~~the~~ a capability key secret key (see 3.1.40)~~.~~ as follows:

a) If the CDB CONTINUATION LENGTH field (see 5.2.1) contains zero (i.e., if there is no CDB continuation segment), then contents of the SOLO CREDENTIAL INTEGRITY CHECK VALUE field are the only capability key used to validate all aspects of the command; or
b) If the CDB CONTINUATION LENGTH field contains a non-zero value, then:
   A) If the CDB continuation segment (see 5.x) contains an extension capabilities CDB continuation descriptor (see 5.y.z), then the integrity check values in the credential are used as follows:
      a) The contents of the SOLO CREDENTIAL INTEGRITY CHECK VALUE field are the capability key that is used to validate the CDB; and
      b) The contents of the EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field are the capability key that is used to validate all other aspects of the command;

   or
   B) If the CDB continuation segment does not contain an extension capabilities CDB continuation descriptor, then the contents of the SOLO CREDENTIAL INTEGRITY CHECK VALUE field are the capability key that is used to validate all aspects of the command.

~~For application clients, the capability key is the contents of the CREDENTIAL INTEGRITY CHECK VALUE field (see 4.12.5.1).~~

The device server processing of each command relies on only the capability portion or portions of the credential (see 4.12.5.1) that the application client has copied into the CDB and CDB continuation segment (see 5.x). Since the capability or capabilities do ~~does~~ not include the ~~CREDENTIAL INTEGRITY CHECK VALUE field~~ integrity check value or values from the credential, the device server needs to compute the capability key or keys for each processed command by:

~~1) Reconstructing the credential containing the CDB capability as described in 4.12.6.2; and~~
~~2) Computing the credential integrity check value for the reconstructed credential using the algorithm, inputs, and secret key specified in 4.12.6.3.~~
1) Constructing a credential that contains only the solo capability as described in 4.12.6.2.1;
2) Computing the solo integrity check value capability key for the constructed credential using the algorithm, inputs, and secret key specified in 4.12.6.3.1; and
3) If an extension capabilities CDB continuation descriptor (see 5.y.z) appears in the CDB continuation segment (see 5.x), if any, then:
   1) Adding the extension capabilities to the credential constructed in step 1) as described in 4.12.6.2.2; and

2) Computing the extended integrity check value capability key for the constructed credential using the algorithm, inputs, and secret key specified in 4.12.6.3.2.

NOTE 3 ~~The two steps used by the device server to compute capability key are the first two steps that the device server uses to validate a credential (see 4.12.6.1).~~ The device server may perform the capability key computations steps described in this subclause ~~these two steps only~~ once for every command processed or repeat them every time a capability key is needed for a validation operation.

{{Subclauses reordered at this point to focus on the credential changes needed to support associating more than one capability with a command.}}

4.12.3 Preparing credentials

4.12.3.1 Overview

{{All the text shown as unchanged or deleted in this subclause appears in 4.12.3 in OSD-2 r03. This is done to simplify reviewing the proposed changes.}}

In response to a request from an application client, the security manager shall prepare and return a credential as follows:

1) Forward the access requests from the application client to the policy/storage manager. If the policy/storage manager denies the forwarded ~~request~~ requests an error shall be returned to the requesting application client;
2) Insert ~~the capability~~ one of the capabilities returned by the policy/storage manager as the solo capability in the credential;
3) Set the credential OSD SYSTEM ID field to the value in the OSD system ID attribute in the Root Information attributes page (see 7.1.2.8) of the OSD logical unit to which the credential applies;
4) Setup the solo capability as described in 4.12.3.2;
5) Unless the SECURITY METHOD field in the solo capability contains NOSEC, compute the solo credential integrity check value as described in 4.12.6.3.1 and place the result in the SOLO CREDENTIAL INTEGRITY CHECK VALUE field;
6) If more than one capability was requested from and returned by the policy/storage manager, then:
    1) Insert each capability that is not the solo capability as an extension capability;
    2) Setup each extension capability as described in 4.12.3.2;
    3) Compute the total bytes of extension capabilities and place this value in the EXTENSION CAPABILITIES LENGTH field; and
    4) Unless the SECURITY METHOD field in the solo capability contains NOSEC, compute the extended credential integrity check value as described in 4.12.6.3.2 and place the result in the EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field;
7) If only one capability was requested from and returned by the policy/storage manager, set the EXTENSION CAPABILITIES LENGTH field to zero;
8) If the SECURITY METHOD field in the solo capability contains NOSEC, set the SOLO CREDENTIAL INTEGRITY CHECK VALUE field and the EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field to zero; and
9) Return the credential thus constructed to the application client.
~~5) If the SECURITY METHOD field contains NOSEC, place zero in the CREDENTIAL INTEGRITY CHECK VALUE field and return the credential to the application client;~~
~~6) Otherwise:~~
… (see 4.12.3.2)
~~10) Compute the credential integrity check value as described in 4.12.6.3, placing the result in the CREDENTIAL INTEGRITY CHECK VALUE field in the credential; and~~
~~11) Return the credential thus constructed to the application client with the credential integrity check value serving as the capability key.~~

4.12.3.2 Capability setup steps for credential preparation

{{All the text shown as unchanged or deleted in this subclause appears in 4.12.3 in OSD-2 r03. This is done to simplify reviewing the proposed changes.}}

For each capability (see 4.11.2) in a credential, the security manager shall setup the capability as follows:

1) Set the capability SECURITY METHOD field as follows:
   A) Select a security method other than the partition default:
      a) If the application client requested use of a specific security method, and use of the requested security method is allowed by both the addressed partition and the maintained security policy information, set the capability SECURITY METHOD field to the requested value;
      b) If the maintained security policy information requires use of a specific security method for the requesting application client, set the capability SECURITY METHOD field to that value;

   or

   B) Use the partition default:
      a) If the application client requested a credential to be used in a SET KEY command (see 6.29) or a SET MASTER KEY command (see 6.30), set the capability SECURITY METHOD field to the value in the default security method attribute in the Root Policy/Security attributes page (see 7.1.2.21);
      b) If the capability OBJECT TYPE field contains ROOT, set the capability SECURITY METHOD field to the value in the default security method attribute in the Root Policy/Security attributes page;
      c) If the capability OBJECT TYPE field contains PARTITION, set the capability SECURITY METHOD field to the value in the default security method attribute in the Partition Policy/Security attributes page (see 7.1.2.22) for partition zero (see 4.6.4);
      d) Otherwise, set the capability SECURITY METHOD field to the value in the default security method attribute in the Partition Policy/Security attributes page for the partition whose Partition ID is contained in the capability ALLOWED PARTITION_ID field;
2) If the SECURITY METHOD field contains NOSEC, no additional capability setup is needed (i.e., exit the processing described in this subclause) ~~place zero in the credential integrity check value field and return the credential to the application client~~;
3) Otherwise:
4) Set the capability KEY VERSION field to the number of the working key secret key used to compute the ~~credential~~ applicable integrity check value in the credential. If a secret key other than a working key is used by the integrity check value computation ~~to compute the credential integrity check value~~ (e.g., {{note added comma}} for a SET KEY command (see 6.29) or a SET MASTER KEY command (see 6.30)), then set the capability KEY VERSION field to zero;
5) Set the capability INTEGRITY CHECK VALUE ALGORITHM field to the low order four bits of the attribute number of the attribute in the Root Policy/Security attributes page (see 7.1.2.21) that indicates the algorithm used to compute all integrity check values related to this capability ~~credential~~ (e.g., if attribute number 8000 0003h identifies the integrity check value algorithm used in this capability ~~credential~~, then the INTEGRITY CHECK VALUE ALGORITHM field shall contain three); and
6) As specified by the maintained security policy information, modify other capability fields, including but not limited to the following:
   A) Setting the CAPABILITY EXPIRATION TIME field to a value that is consistent with the policy;
   B) Ensuring that the capability AUDIT field and CAPABILITY DISCRIMINATOR field contain non-zero values;
   C) Setting the capability OBJECT CREATED TIME field to a non-zero value that is consistent with 4.11.2.2.1 usage; and
   D) Ensuring that the POL/SEC bit in the PERMISSIONS BIT MASK field is set to zero, if appropriate~~;~~.

…

4.12.6 OSD device server security algorithms

4.12.6.1 Credential validation

4.12.6.1.1 Overview

The processes described in this subclause do not apply if the CDB SECURITY METHOD field specifies the NOSEC security method (i.e., if the CDB SECURITY METHOD field contains zero).

If the CDB SECURITY METHOD field specifies the CMDRSP security method or the ALLDATA security method, the device server shall validate the CDB REQUEST NONCE field as described in 4.12.7.2.

The device server validates a credential by verifying the integrity check values in the CDB and CDB continuation segment (see 5.x), if any, that the application client computed using one or both of the capability keys (see 4.12.5.2) in the credential.

The device server shall validate the solo portion of a credential (see 4.12.5) ~~associated with a CDB by:~~ as described in 4.12.6.1.2.

If the CDB CONTINUATION LENGTH field (see 5.2.1) contains a non-zero value, then:

a) If the CDB continuation segment (see 5.x) contains an extension capabilities CDB continuation descriptor (see 5.y.z), then the device server shall validate the extension portion of a credential as described in 4.12.6.1.3; or
b) If the CDB continuation segment does not contain an extension capabilities CDB continuation descriptor, then the device server shall revalidate the solo portion of a credential as described in 4.12.6.1.4.

{{The following paragraph appears at the end of the 4.12.6.1 in OSD-2 r03.}}

If the validation of any portion of a credential results in a CHECK CONDITION status being returned, the state of the OSD objects and attributes shall not be altered in any detectable way.

4.12.6.1.2 Validating the solo portion of a credential

{{All the text shown as unchanged or deleted in this subclause appears in 4.12.6.1 in OSD-2 r03. This is done to simplify reviewing the proposed changes.}}

The device server shall validate the solo portion of a credential (see 4.12.5) ~~associated with a CDB~~ by:

~~1) Reconstructing the credential containing the capability as described in 4.12.6.2;~~
~~2) Computing the credential integrity check value for the reconstructed credential using the algorithm, inputs, and secret key specified in 4.12.6.3;~~
1) Constructing a credential that contains only the solo capability as described in 4.12.6.2.1;
2) Computing the solo integrity check value capability key for the constructed credential using the algorithm, inputs, and secret key specified in 4.12.6.3.1;
3) Computing the request integrity check value using:
   A) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the capability INTEGRITY CHECK VALUE ALGORITHM field of the solo capability (see 4.12.3);
   B) Based on the contents of the CDB SECURITY METHOD field, one of the following arrays of bytes:
      a) For the CAPKEY security method, the security token (see 4.12.4.3); or
      b) For the CMDRSP security method and the ALLDATA security method, all the bytes in the CDB with the bytes in the REQUEST INTEGRITY CHECK VALUE field set to zero;

   and

C) The ~~credential integrity check value~~ solo integrity check value capability key computed in step 2) as the secret key;

and

4) Verifying that the request integrity check value matches the contents of the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8). If the contents in the REQUEST INTEGRITY CHECK VALUE field in the CDB do not match the computed solo integrity check value, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB. {{Note addition of missing period.}}

4.12.6.1.3 Validating the extension portion of a credential

If the CDB continuation segment (see 5.x) contains an extension capabilities CDB continuation descriptor (see 5.y.z), then the device server shall validate the extension portion of a credential (see 4.12.5) by:

1) Adding extension capability information to the credential constructed during the validation of the solo portion of the credential (see 4.12.6.1.2) as described in 4.12.6.2.2;
2) Computing the extended integrity check value capability key for the constructed and extended credential using the algorithm, inputs, and secret key specified in 4.12.6.3.2;
3) Computing the continuation integrity check value using:
   A) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the capability INTEGRITY CHECK VALUE ALGORITHM field of the solo capability (see 4.12.3);
   B) Based on the contents of the CDB SECURITY METHOD field, one of the following arrays of bytes:
      a) For the CAPKEY security method, the security token (see 4.12.4.3); or
      b) For the CMDRSP security method and the ALLDATA security method, the following array of bytes:
         1) All the bytes in the CDB REQUEST INTEGRITY CHECK VALUE field; and
         2) All the bytes in the CDB continuation segment (see 5.x) with the bytes in the CONTINUATION INTEGRITY CHECK VALUE field set to zero;

      and
   C) The extended integrity check value capability key computed in step 2) as the secret key;

   and
4) Verifying that the continuation integrity check value matches the contents of the CONTINUATION INTEGRITY CHECK VALUE field in the CDB continuation segment (see 5.x). If the contents in the CONTINUATION INTEGRITY CHECK VALUE field in the CDB do not match the computed extension integrity check value, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

4.12.6.1.4 Validating a CDB continuation segment using the solo portion of a credential

If the CDB CONTINUATION LENGTH field (see 5.2.1) contains a non-zero value but the CDB continuation segment (see 5.x) does not contain an extension capabilities CDB continuation descriptor (see 5.y.z), then the device server shall validate the CDB continuation segment using the credential's solo portion by:

1) Locating or reconstructing the credential constructed in 4.12.6.1.2 that contains only the solo capability and the associated solo integrity check value;
2) Computing the continuation integrity check value using:
   A) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the capability INTEGRITY CHECK VALUE ALGORITHM field of the solo capability (see 4.12.3);
   B) Based on the contents of the CDB SECURITY METHOD field, one of the following arrays of bytes:
      a) For the CAPKEY security method, the security token (see 4.12.4.3); or
      b) For the CMDRSP security method and the ALLDATA security method, the following array of bytes:
         1) All the bytes in the CDB REQUEST INTEGRITY CHECK VALUE field; and

2) All the bytes in the CDB continuation segment (see 5.x) with the bytes in the CONTINUATION INTEGRITY CHECK VALUE field set to zero;

and

C) The solo integrity check value capability key computed in step 1) as the secret key;

and

3) Verifying that the continuation integrity check value matches the contents of the CONTINUATION INTEGRITY CHECK VALUE field in the CDB continuation segment (see 5.x). If the contents in the CONTINUATION INTEGRITY CHECK VALUE field in the CDB do not match the computed extension integrity check value, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB.

4.12.6.2 Reconstructing the credential

4.12.6.2.1 Reconstructing the solo portion of a credential

{{All the text shown as unchanged or deleted in this subclause appears in 4.12.6.2 in OSD-2 r03. This is done to simplify reviewing the proposed changes.}}

The device server reconstructs the solo portion of a credential from a CDB capability by:

1) Copying the value in the OSD system ID attribute in the Root Information attributes page (see 7.1.2.8) to the OSD SYSTEM ID field of the reconstructed credential; ~~and~~
2) Copying the capability from the CDB to the solo capability portion of reconstructed credential~~.~~; and
3) Setting the EXTENSION CAPABILITIES LENGTH field to zero.

{{There are no ordering requirements needed for proper operation of the above list. It should be changed to an unordered list.}}

The SOLO CREDENTIAL INTEGRITY CHECK VALUE field and EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field are ~~is~~ not used in a reconstructed credential and are set to zero.

4.12.6.2.2 Reconstructing the extension portion of a credential

Using the contents of the reconstructed solo portion of a reconstructed credential (see 4.12.6.2.1), the device server reconstructs the extension portion of a credential from the contents of an extension capabilities CDB continuation descriptor (see 5.y.z) by:

a) Not modifying the contents of the solo capability portion, OSD SYSTEM ID field, and SOLO CREDENTIAL INTEGRITY CHECK VALUE field in the input reconstructed credential;
b) Setting the EXTENSION CAPABILITIES LENGTH field in the reconstructed credential to the value in the CONTINUATION DESCRIPTOR LENGTH field of the extension capabilities CDB continuation descriptor minus four; and
c) Copying all of the bytes in all of the extension capabilities in the extension capabilities CDB continuation descriptor to the extension capabilities portion of the reconstructed credential.

The SOLO CREDENTIAL INTEGRITY CHECK VALUE field and EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field are not used in a reconstructed credential and are set to zero.

4.12.6.3 Computing the integrity check values for a credential ~~integrity check value~~

4.12.6.3.1 Computing the solo integrity check value for a credential

{{All the text shown as unchanged or deleted in this subclause appears in 4.12.6.3 in OSD-2 r03. This is done to simplify reviewing the proposed changes.}}

The solo credential integrity check value shall be computed using:

a) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the ~~capability~~ INTEGRITY CHECK VALUE ALGORITHM field in the solo capability in the credential (see 4.12.3);
b) All the bytes in the solo capability portion of the credential (see 4.12.5.1) and the OSD SYSTEM ID field of the credential; and
~~b) The following bytes:~~
~~A) All of the bytes in all of the fields defined for the credential (see 4.12.5.1);~~
~~B) Except the bytes in the CREDENTIAL INTEGRITY CHECK VALUE field;~~
~~and~~
c) The secret key selected as follows;
   A) If the OBJECT TYPE field in the credential's (see 4.12.5.1) solo capability (see 4.11.2.2) contains COLLECTION or USER, the secret key is the authentication working key:
      a) Identified by the KEY VERSION field in the credential's (see 4.12.5.1) solo capability; and
      b) Associated with the partition identified by the ~~PARTITION_ID field in the CDB~~ ALLOWED PARTITION_ID field in the credential's (see 4.12.5.1) solo capability;
   B) If the OBJECT TYPE field in the credential's solo capability contains ROOT or PARTITION and the command is not SET KEY and not SET MASTER KEY, the secret key is the authentication working key for partition zero identified by the KEY VERSION field in the credential's (see 4.12.5.1) solo capability;
   C) If the command is SET KEY (see 6.29), the secret key that is selected as follows:
      a) If the KEY TO SET field in the CDB contains 01b (i.e., update root key), the authentication master key;
      b) If the KEY TO SET field in the CDB contains 10b (i.e., update partition key), the authentication root key; or
      c) If the KEY TO SET field in the CDB contains 11b (i.e., update working key), the authentication partition key for the partition identified by the PARTITION_ID field in the CDB;

      or
   D) For the SET MASTER KEY command:
      a) For the SEED EXCHANGE step (see 6.30.2), the authentication master key; or
      b) For the CHANGE MASTER KEY step (see 6.30.3), the next authentication master key computed after GOOD status has been returned by the SEED EXCHANGE step (see 6.30.2).

4.12.6.3.2 Computing the extended integrity check value for a credential

The extended credential integrity check value shall be computed as follows:

1) An intermediate integrity check value shall be computed using:
   A) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the INTEGRITY CHECK VALUE ALGORITHM field in the solo capability in the credential (see 4.12.3);
   B) All the bytes in the credential (see 4.12.5.1) except the EXTENDED CREDENTIAL INTEGRITY CHECK VALUE field; and
   C) The secret key selected as follows;
      a) If the OBJECT TYPE field in the credential's (see 4.12.5.1) solo capability (see 4.11.2.2) contains COLLECTION or USER, the secret key is the authentication working key:
         A) Identified by the KEY VERSION field in the credential's (see 4.12.5.1) solo capability; and

B) Associated with the partition identified by the ALLOWED PARTITION_ID field in the credential's (see 4.12.5.1) solo capability;

or

b) If the OBJECT TYPE field in the credential's (see 4.12.5.1) solo capability contains ROOT or PARTITION, the secret key is the authentication working key for partition zero identified by the KEY VERSION field in the credential's (see 4.12.5.1) solo capability;

and

2) Each extension capability in the credential (see 4.12.5.1) shall be processed in the order in which it appears in the credential, and a new intermediate integrity check value shall be computed based on the extension capability being processed using:
   A) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the INTEGRITY CHECK VALUE ALGORITHM field in the solo capability in the credential (see 4.12.3);
   B) All the bytes in the in the previously computed intermediate integrity check value; and
   C) The secret key selected as follows;
      a) If the OBJECT TYPE field in the credential's (see 4.12.5.1) extension capability (see 4.11.2.2) being processed contains COLLECTION or USER, the secret key is the authentication working key:
         A) Identified by the KEY VERSION field in the credential's (see 4.12.5.1) extension capability being processed; and
         B) Associated with the partition identified by the ALLOWED PARTITION_ID field in the credential's (see 4.12.5.1) extension capability being processed;

         or
      b) If the OBJECT TYPE field in the credential's extension capability being processed contains ROOT or PARTITION, the secret key is the authentication working key for partition zero identified by the KEY VERSION field in the credential's (see 4.12.5.1) extension capability being processed.

The extended credential integrity check value is the last intermediate integrity check value computed.

4.12.6.4 Invalidating credentials

The security manager may invalidate the credentials for one OSD object by requesting that the policy/storage manager change the policy access tag attribute in the policy/security attributes page associated with that OSD object (see 4.11.3.2) or objects to a value other than the policy access tag value that is contained in the credential's capability or capabilities.

The security manager may invalidate credentials for an entire partition by using the SET KEY command (see 6.29) to update the working key version used to compute the ~~credential~~ integrity check values ~~value~~ in those credentials.

…

{{Subclauses reordered at this point to focus on the security method changes needed to support associating more than one capability with a command.}}

{{The following text describing security methods were in changes 5 and 6 in 08-105r1. They are part of the credential changes now because the modifications involve (among other things) which capability key to use.}}}

4.12.4.3 The CAPKEY security method

The CAPKEY security method validates the integrity of the capability information in each CDB and in the CDB continuation segment (see 5.x), if any.

The application client computes the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8) contents using:

a) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field (see 4.12.3);
b) The security token returned in the Security Token VPD page (see 7.5.3); and
c) The ~~credential~~ capability key (see 4.12.5.2) for the solo capability.

If the CDB CONTINUATION LENGTH field (see 5.2.1) is not set to zero, the application client computes the CONTINUATION INTEGRITY CHECK VALUE field contents in the CDB continuation segment (see 5.x) using:

a) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field (see 4.12.3);
b) The security token returned in the Security Token VPD page (see 7.5.3); and
c) The applicable capability key (see 4.12.5.2).

The device server validates the credential as described in 4.12.6.1.

The CAPKEY security method is useful when the service delivery subsystem between the OSD device server and application client is secured via methods specified in the applicable SCSI transport protocol, with both the CAPKEY security method and SCSI transport protocol secure channel contributing to securing communications as shown in table 28 (see 4.12.4.1).

4.12.4.4 The CMDRSP security method

The CMDRSP security method validates the integrity of the CDB, the CDB continuation segment, if any, status, and sense data for each command.

The application client computes the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8) contents using:

a) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field (see 4.12.3);
b) All the bytes in the CDB with the bytes in the REQUEST INTEGRITY CHECK VALUE field set to zero; and
c) The ~~credential~~ capability key (see 4.12.5.2) for the solo capability.

If the CDB CONTINUATION LENGTH field (see 5.2.1) is not set to zero, the application client computes the CONTINUATION INTEGRITY CHECK VALUE field contents in the CDB continuation segment (see 5.x) using:

a) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field (see 4.12.3);
b) The following array of bytes:
    1) All the bytes in the CDB REQUEST INTEGRITY CHECK VALUE field; and
    2) All the bytes in the CDB continuation segment (see 5.x) with the bytes in the CONTINUATION INTEGRITY CHECK VALUE field set to zero;

    and
c) The applicable capability key (see 4.12.5.2).

The device server validates the credential as described in 4.12.6.1.

If the credential validation process ~~successfully~~ validates the request integrity check value without errors and the continuation integrity check value, if any, is validated without errors ~~associated with the command~~, then the device server shall:

1) Compute an integrity check value for the response data using:

A) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field (see 4.12.3);
B) The following array of bytes:
1) The request nonce from the CDB (see 5.2.8);
2) The status byte; and
3) If the status is CHECK CONDITION, the sense data with the RESPONSE INTEGRITY CHECK VALUE field in the OSD response integrity check value sense data descriptor (see 4.15.2.2) set to zero;
and
C) The applicable capability key (see 4.12.5.2) for the reconstructed credential (see 4.12.6.2);
and
2) Place the computed integrity check value in the following location:
A) If the status is not CHECK CONDITION, the computed integrity check value shall be placed in the response integrity check value attribute in the Current Command attributes page (see 7.1.2.29); or
B) If the status is CHECK CONDITION, the computed integrity check value shall be placed in the RESPONSE INTEGRITY CHECK VALUE field in the OSD response integrity check value sense data descriptor (see 4.15.2.2) in the sense data.

…

4.12.4.5 The ALLDATA security method

The ALLDATA security method validates the integrity of all data in transit between an application client and device server.

The application client computes the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8) contents using the same algorithm specified for the CMDRSP security method (see 4.12.4.4). If the CDB CONTINUATION LENGTH field (see 5.2.1) is not set to zero, application client computes the CDB CONTINUATION INTEGRITY CHECK VALUE field (see 5.x) contains using the same algorithm specified for the CMDRSP security method (see 4.12.4.4)

The device server validates the credential as described in 4.12.6.1.

The application client also computes the data-out integrity check value using:

a) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field (see 4.12.3);
b) The following array of bytes:
1) All the bytes in the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8); and
2) The used bytes in the following Data-Out Buffer segments (see 4.14.4):
1) Command data ~~or parameter data~~;
2) Set attributes; and
3) Get attributes;
and
c) The ~~credential~~ applicable capability key (see 4.12.5.2).

…

The application client places the data-out integrity information (see table 29) in the Data-Out Buffer starting at the byte specified by the CDB DATA-OUT INTEGRITY CHECK VALUE OFFSET field (see 5.2.8).

Table 29 — Data-out integrity information format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			NUMBER OF COMMAND ~~OR PARAMETER~~ DATA BYTES				
7								(LSB)
8	(MSB)			NUMBER OF SET ATTRIBUTES BYTES				
15								(LSB)
16	(MSB)			NUMBER OF GET ATTRIBUTES BYTES				
23								(LSB)
24	(MSB)			DATA-OUT INTEGRITY CHECK VALUE				
55~~43~~								(LSB)

The NUMBER OF COMMAND ~~OR PARAMETER~~ DATA BYTES field specifies the number of bytes from the command data ~~or parameter data~~ segment that are included in the data-out integrity check value. If the value in the CDB LENGTH field, if any, ~~or the value in the CDB PARAMETER LIST LENGTH field, if any,~~ is larger than the value in the NUMBER OF COMMAND ~~OR PARAMETER~~ DATA BYTES field, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

…

The device server shall validate the data-out integrity check value by:

1) Computing an integrity check value using:
   A) The algorithm indicated by the attribute in the Root Policy/Security attributes page (see 7.1.2.21) whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field (see 4.12.3);
   B) The following array of bytes:
      1) All the bytes in the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8); and
      2) The following bytes from Data-Out Buffer:
         1) The number of bytes specified by the NUMBER OF COMMAND ~~OR PARAMETER~~ DATA BYTES field starting at the Data-Out Buffer byte ~~offset zero~~ that follows the CDB continuation segment (see 4.14.4) (i.e., the byte offset specified by the contents of the CDB CONTINUATION LENGTH field (see 5.2.1);
         2) The number of bytes specified by the NUMBER OF SET ATTRIBUTES BYTES field starting at the Data-Out Buffer byte offset specified by the CDB SET ATTRIBUTES LIST OFFSET field (see 5.2.4.4); and
         3) The number of bytes specified by the NUMBER OF GET ATTRIBUTES BYTES field starting at the Data-Out Buffer byte offset specified by the CDB GET ATTRIBUTES LIST OFFSET field (see 5.2.4.4);

      and
   C) The applicable capability key (see 4.12.5.2) for the reconstructed credential (see 4.12.6.2); and
2) Comparing the results to contents of the DATA-OUT INTEGRITY CHECK VALUE field.

…

The device server shall compute the data-in integrity check value using:

a) The algorithm indicated by the attribute in the Root Policy/Security attributes page whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field;
b) The following array of bytes:
   1) All the bytes in the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8); and
   2) The used bytes in the following Data-In Buffer segments (see 4.14.3):
      1) Command data or parameter data; and
      2) Retrieved attributes;

   and
c) The applicable capability key (see 4.12.5.2) for the reconstructed credential (see 4.12.6.2).

…

After status has been received, the application client validates the data-in integrity check value by:

1) Computing an integrity check value using:
   A) The algorithm indicated by the attribute in the Root Policy/Security attributes page whose attribute number is specified in the solo capability INTEGRITY CHECK VALUE ALGORITHM field;
   B) The following array of bytes:
      1) All the bytes in the CDB REQUEST INTEGRITY CHECK VALUE field (see 5.2.8); and
      2) The following bytes from Data-In Buffer:
         1) The number of bytes specified by the NUMBER OF COMMAND OR PARAMETER BYTES field starting at the Data-In Buffer byte offset zero; and
         2) The number of bytes specified by the NUMBER OF RETRIEVED ATTRIBUTES BYTES field starting at the Data-In Buffer byte offset specified by the CDB RETRIEVED ATTRIBUTES OFFSET field (see 5.2.4);

      and
   C) The ~~credential~~ applicable capability key (see 4.12.5.2);

   and
2) Comparing the results to contents of the DATA-IN INTEGRITY CHECK VALUE field.

…

5.2 Fields commonly used in OSD commands

5.2.1 Overview

…

Table 49 — OSD service action specific fields

Bit / Byte	7	6	5	4	3	2	1	0
10	Reserved			DPO [a]	FUA [a]	ISOLATION (see 5.2.5)		
… 79	… …							
80 183	Capability (see 5.2.c) ~~(see 4.11.2.2)~~							
…	…							

{{Every CDB definition in Clause 6 must be changed as shown in table 49 above.}}

…

{{Insert 5.2.c in the proper alphabetical order among the 5.2.? subclauses.}}

5.2.c Capability

The capability is defined in 4.11.2.2. Any security method other than NOSEC (see 4.12.4) is design to return an error unless the capability in the CDB is a solo capability (see 4.12.5).

…

{{The text describing every CDB definition in Clause 6 must be changed as follows.}}

The capability is defined in ~~4.11.2.2~~ 5.2.c.

…

5.y.1 Overview

…

The CDB CONTINUATION DESCRIPTOR TYPE field (see table x4) specifies the format of the CDB continuation descriptor type specific data.

Table x4 — CDB CONTINUATION DESCRIPTOR TYPE field

Value	Description	Reference
0000h	No more continuation descriptors [a]	
FFEEh	Extension capabilities	5.y.z
{{Other rows of this table are filled in by other changes in this proposal.}} {{The most complete body for this table can be found in change 7 table x4.}}		
all other values	Reserved	

[a] Since the CDB continuation segment pad bytes, if any, are set to zero (see 5.x), encountering a CDB continuation descriptor type of zero shall be processed in the same way as reaching the last byte of the CDB continuation segment.

…

5.y.z Extension capabilities CDB continuation descriptors

{{All of 5.y.z is new. The use of change markups is suspended for the remainder of 5.y.z.}}

The extension capabilities CDB continuation descriptor (see table x5) adds one or more capabilities to the set of capabilities associated with the command.

Table x5 — Extension capabilities CDB continuation descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
	CDB continuation descriptor header							
0	(MSB)			CDB CONTINUATION DESCRIPTOR TYPE (FFEEh)				
1								(LSB)
2	Reserved							
3	Reserved					PAD LENGTH (000b)		
4	(MSB)			CONTINUATION DESCRIPTOR LENGTH (k-7)				
7								(LSB)
	CDB continuation descriptor type specific data							
8	Extension capability (see 4.11.2.2) [first]							
111								
	⋮							
k-103	Extension capability (see 4.11.2.2) [last]							
k								

The CDB CONTINUATION DESCRIPTOR TYPE field contains FFEEh (i.e., extension capabilities CDB continuation descriptor).

The PAD LENGTH field is set to zero to indicate that no pad bytes are needed to eight byte align an extension capabilities CDB continuation descriptor. If the PAD LENGTH field is not set to zero in an extension capabilities CDB continuation descriptor, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The CONTINUATION DESCRIPTOR LENGTH field contains the number of bytes that follow in this descriptor.

Each extension capability is a capability (see 4.11.2.2) that adds to the object access capabilities of the command.

Unless the SECURITY METHOD field in the CDB specifies the NOSEC security method (i.e., if the CDB SECURITY METHOD field contains zero), the contents of the extension capabilities CDB continuation descriptor should be copied from a credential (see 4.12.5). The device server shall validate this credential as described in 4.12.6.1.

If a CDB continuation segment (see 5.x) contains more than one extension capabilities CDB continuation descriptor, the command shall be terminated with CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN PARAMETER LIST.

{{Subclauses reordered at this point to focus on the capability definition changes needed to support associating more than one capability with a command.}}

4.11.2 Capabilities

4.11.2.1 Introduction

Each CDB defined by this standard includes a solo capability (see 4.12.5.1) ~~(see 4.11.2.2)~~ whose contents specify the command functions (see 3.1.10) that the device server is allowed to process in response to the command. If allowed by a command's definition, the CDB continuation segment (see 5.x) may be used to add one or more extension capabilities (see 5.y.z) to the command processing inputs.

The device server validates that the requested command functions are allowed by ~~the~~ a solo capability or an extension capability based on:

a) The type of functions (e.g., read, write, attributes setting, attributes retrieval); and
b) The OSD object or objects on which the command functions are to be processed.

The policies that determine which capabilities are provided to which application clients are outside the scope of this standard.

…

4.11.2.2 Capability format

4.11.2.2.1 Introduction

A capability (see table 12) is included in a CDB to enable the device server to verify that the sender is allowed to perform the command functions (see 3.1.10) described by the CDB.

Table 12 — Capability format

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved				CAPABILITY FORMAT (2h)			
1	KEY VERSION				INTEGRITY CHECK VALUE ALGORITHM			
2	Reserved				SECURITY METHOD			
3	Reserved							
4	(MSB)	CAPABILITY EXPIRATION TIME						
9								(LSB)
10		AUDIT						
29								
30	(MSB)	CAPABILITY DISCRIMINATOR						
41								(LSB)
42	(MSB)	OBJECT CREATED TIME						
47								(LSB)
48	OBJECT TYPE							
49		PERMISSIONS BIT MASK						
53								
54	Reserved							
55	OBJECT DESCRIPTOR TYPE				Reserved			
56	(MSB)	ALLOWED ATTRIBUTES ACCESS						
59								(LSB)
60		OBJECT DESCRIPTOR						
103								

The CAPABILITY FORMAT field (see table 13) specifies the format of the capability. If capabilities are coordinated with the security manager, the capability format also is the credential format. The policy/storage manager shall set the CAPABILITY FORMAT field to ~~1h~~ 2h (i.e., the format defined by this standard).

Table 13 — Capability format values

Value	Description
0h	No capability
1h	Obsolete
2h	The format defined by this standard
3h - Fh	Reserved

If the CAPABILITY FORMAT field contains 2h, the device server shall verify that the command functions requested by a CDB and any CDB continuation descriptors (see 5.x) are permitted by ~~the capability~~ at least one of the capabilities associated with the command (i.e., by the capability in the CDB (see 5.2.1) or by a capability in the extension capabilities CDB continuation descriptor (see 5.y.z)) as described in this subclause. The device server may verify that a command function is permitted after other command functions are completed. The device server shall verify that a command function is permitted before any part of the command function is performed. (E.g., the device server may delay verifying that the set attributes command functions specified by a set attributes list are allowed until the requested read command function is completed, but all the capability permissions concerning the setting attributes are to be verified before any attribute values are changed.)

The KEY VERSION field, INTEGRITY CHECK VALUE ALGORITHM field, and SECURITY METHOD field are used by the security manager (see 4.12.3). If capabilities are not coordinated with the security manager, the KEY VERSION field, INTEGRITY CHECK VALUE ALGORITHM field, and SECURITY METHOD field are reserved.

If CDB contains a non-zero value in the SECURITY METHOD field, the integrity of the CDB and CDB continuation segment, if any, shall be validated (see 4.12.6.1) before any other command processing actions are undertaken (i.e., before verifying that command functions requested in the CDB are permitted by the capability).

The command shall be terminated as described in 4.11.2.2.n ~~with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB~~, if the CDB SECURITY METHOD field or CAPABILITY FORMAT field contains zero and one of the following is true:

a) The command is SET KEY (see 6.29) or SET MASTER KEY (see 6.30); or
b) The default security method attribute in the attributes page that is located as follows based on the contents of the OBJECT TYPE field specifies a default security method other than NOSEC:
   A) If the capability OBJECT TYPE field contains ROOT, the default security method attribute in the Root Policy/Security attributes page (see 7.1.2.21);
   B) If the capability OBJECT TYPE field contains PARTITION, the default security method attribute in the Partition Policy/Security attributes page (see 7.1.2.22) for partition zero (see 4.6.4); or
   C) If the capability OBJECT TYPE field contains COLLECTION or USER, the default security method attribute in the Partition Policy/Security attributes page for the partition whose Partition ID is contained in the capability ALLOWED PARTITION_ID field.

The CAPABILITY EXPIRATION TIME field specifies the value of the clock attribute in the Root Information attributes page (see 7.1.2.8) after which this capability is no longer valid. If a ~~CDB~~ CAPABILITY EXPIRATION TIME field contains a value other than zero and the value of the clock attribute in the Root Information attributes page (see 7.1.2.8) is greater than the value in the CAPABILITY EXPIRATION TIME field, the command shall be terminated as described in 4.11.2.2.n ~~with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB~~.

Successful use of the capability expiration time requires some degree of synchronization between the clocks of the device server, policy/storage manager, and security manager. The protocol for synchronizing the clocks is outside the scope of this standard.

The AUDIT field is a vendor specific value that the policy/storage manager and/or security manager may use to associate the capability and credential with a specific application client.

The CAPABILITY DISCRIMINATOR field contains a nonce (see 3.1.24) that differentiates one capability and credential from another.

The OBJECT CREATED TIME field specifies the contents of the created time attribute for the OSD object (see table 14) to which the capability applies. A value of zero specifies that any object created time is allowed.

Table 14 — Created time for OSD objects by type

Object Type (see table 15)	Attributes page containing created time attribute to which the capability OBJECT CREATED TIME field is applies
ROOT	Partition Timestamps attributes page (see 7.1.2.16) for partition zero (see 3.1.33)
PARTITION	Partition Timestamps attributes page
COLLECTION	Collection Timestamps attributes page (see 7.1.2.17)
USER	User Object Timestamps attributes page (see 7.1.2.18)

If a CDB OBJECT CREATED TIME field contains a value other than zero and the value in the OBJECT CREATED TIME field is not identical to the value in the created time attribute from the associated timestamps attributes page (see table 14), then the command shall be terminated as described in 4.11.2.2.n ~~with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB~~.

The OBJECT TYPE field (see table 15) specifies the type of OSD object to which this capability allows access and aids in the determination of how to validate the capability. If capabilities are coordinated with the security manager, the OBJECT TYPE field is used to select the secret key that is used in validating the credential.

Table 15 — Object type values

Value	Name	OSD object type to which access is allowed
01h	ROOT	Root object
02h	PARTITION	Partition
40h	COLLECTION	Collection
80h	USER	User objects
all other values	Reserved	

If the command functions specified by the CDB and CDB continuation segment, if any, are not allowed for the OSD object type specified in the ~~CDB~~ OBJECT TYPE field of any capability associated with the command (i.e., the capability in the CDB (see 5.2.1) and the capabilities, if any, in the CDB continuation segment (see 5.x)), the command shall be terminated as described in 4.11.2.2.n ~~with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB~~.

The PERMISSIONS BIT MASK field (see table 16) specifies which functions are allowed by this capability. More than one permissions bit may be set within the constraints specified in 4.11.2.3 resulting in a single capability that allows more than one command function.

Table 16 — Permissions bit mask format

Bit / Byte	7	6	5	4	3	2	1	0
49	READ	WRITE	GET_ATTR	SET_ATTR	CREATE	REMOVE	OBJ_MGMT	APPEND
50	DEV_MGMT	GLOBAL	POL/SEC	M_OBJECT	QUERY	Reserved		
51	Reserved							
52	Reserved							
53	Reserved							

A READ bit set to one allows read access to the data in an OSD object, but not to the attributes. For the root object, partitions, and collections the data in the OSD object is the list of other objects contained in the OSD object. A READ bit set to zero prohibits read access to the data in an OSD object.

A WRITE bit set to one allows processing of the WRITE command (see 6.32) or an equivalent command, but not access to user object attributes. A WRITE bit set to zero prohibits processing of the WRITE command or an equivalent command.

A GET_ATTR (get attributes) bit set to one allows retrieval of (i.e., read access to) the attributes associated with an OSD object. A GET_ATTR bit set to zero prohibits retrieval of attributes except for the attributes in the Current Command attributes page (see 7.1.2.29).

A SET_ATTR (set attributes) bit set to one allows the setting of (i.e., write access to) the attributes associated with an OSD object except for attributes located in the OSD object's policy/security attributes page (e.g., the User Object Policy/Security attributes page (see 7.1.2.24) if the OSD object is a user object). The setting of attributes located in the OSD object's policy/security attributes page is allowed only if both the SET_ATTR bit and the POL/SEC bit are set to one. A SET_ATTR bit set to zero prohibits the setting of the attributes associated with an OSD object.

A CREATE bit set to one allows the creation of OSD objects. A CREATE bit set to zero prohibits the creation of OSD objects.

A REMOVE bit set to one allows the removal of OSD objects. A REMOVE bit set to zero prohibits the removal of OSD objects.

An OBJ_MGMT (object management) bit set to one allows command functions that may change how the OSD logical unit handles an OSD object without affecting the stored data, stored attributes, commands in the task set, policies, or security for the OSD object. An OBJ_MGMT bit set to zero prohibits such command functions.

An APPEND bit set to one allows processing of the APPEND command (see 6.2), but not access to user object attributes. A APPEND bit set to zero prohibits processing of the APPEND command.

A DEV_MGMT (device management) bit set to one allows command functions that affect the OSD logical unit. A DEV_MGMT bit set to zero prohibits command functions that affect the OSD logical unit.

A GLOBAL bit set to one allows command functions that may affect all the OSD objects in the OSD logical unit. A GLOBAL bit set to zero prohibits command functions that may affect all the OSD objects in the OSD logical unit.

A POL/SEC bit set to one allows command functions that affect the policy/security functions performed for one or more OSD objects. A POL/SEC bit set to zero prohibits command functions that affect the policy/security functions performed for one or more OSD objects.

A multiple objects (M_OBJECT) bit set to one in combination with other permissions bits allows retrieving attributes from multiple user objects, setting attributes in multiple user objects, and removing multiple user objects. An M_OBJECT bit set to zero prohibits multiple user object commands.

A QUERY bit set to one allows searching the user objects in a collection for specified attribute values. An QUERY bit set to zero prohibits searching the user objects in a collection.

The OBJECT DESCRIPTOR TYPE field (see table 17) specifies the format of information that appears in the OBJECT DESCRIPTOR field.

Table 17 — Object descriptor types

Object Descriptor Type	Name	Description	Reference
0h	NONE	The OBJECT DESCRIPTOR field shall be ignored	
1h	USER	A single user object	4.11.2.2.2
2h	PAR	A single partition, including partition zero	4.11.2.2.3
3h	COL	A single collection	4.11.2.2.4
3h - Fh		Reserved	

The ALLOWED ATTRIBUTES ACCESS field (see table 18) places additional restrictions on the attributes that the command is able to access.

Table 18 — ALLOWED ATTRIBUTES ACCESS field

Value	Description
0h	No additional restrictions are placed on attributes accesses.
1h to FFFF FFFEh	The contents of the Attributes Access attributes page attribute for the partition specified by the ALLOWED PARTITION_ID field in the capability object descriptor specified by the ALLOWED ATTRIBUTES ACCESS field restrict the attributes to which access is allowed as described in 7.1.2.20.
FFFF FFFFh	Reserved

If the ALLOWED ATTRIBUTES ACCESS field specifies the attribute number of an attribute that is undefined (see 3.1.50) in the Attributes Access attributes page (see 7.1.2.20) attribute for the partition specified by the ALLOWED PARTITION_ID field in the capability object descriptor, then the command shall be terminated as described in 4.11.2.2.n ~~with a CHECK CONDITION, with the sense key set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB~~.

4.11.2.2.2 USER capability object descriptor

If the object descriptor type is USER (i.e., 1h), the OBJECT DESCRIPTOR field shall have the format shown in table 19, specifying a single user object and a range of bytes with in that user object to which the capability allows access.

Table 19 — User object descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
60	(MSB)			POLICY ACCESS TAG				
63								(LSB)
64	(MSB)			BOOT EPOCH				
65								(LSB)
66				Reserved				
71								
72	(MSB)			ALLOWED PARTITION_ID				
79								(LSB)
80	(MSB)			ALLOWED USER_OBJECT_ID				
87								(LSB)
88	(MSB)			ALLOWED RANGE LENGTH				
95								(LSB)
96	(MSB)			ALLOWED RANGE STARTING BYTE ADDRESS				
103								(LSB)

If the POLICY ACCESS TAG field contains a value other than zero, the policy access tag attribute identified by the command and OBJECT TYPE field (see table 20) is compared to the POLICY ACCESS TAG field contents as part of verifying the capability. If the POLICY ACCESS TAG field contains zero, then no comparison is made to any policy access tag attribute. The policy/storage manager or OSD logical unit changes the policy access tag to prevent unsafe or temporarily undesirable accesses to an OSD object (see 4.11.3.2).

Table 20 — Policy access tag usage for OSD object types and commands

Command	Object Type (see table 15)	Attributes page containing policy access tag attribute to which CDB POLICY ACCESS TAG field is compared
CREATE PARTITION or REMOVE PARTITION	PARTITION	Partition Policy/Security attributes page (see 7.1.2.22) for partition zero (see 3.1.33)
CREATE COLLECTION or REMOVE COLLECTION	COLLECTION	Partition Policy/Security attributes page
CREATE, CREATE AND WRITE, or REMOVE	USER	Partition Policy/Security attributes page
All other commands	ROOT	Partition Policy/Security attributes page for partition zero
	PARTITION	Partition Policy/Security attributes page
	COLLECTION	Collection Policy/Security attributes page (see 7.1.2.23)
	USER	User Object Policy/Security attributes page (see 7.1.2.24)

If the non-zero value in the ~~CDB~~ POLICY ACCESS TAG field is not identical to the value in the policy access tag attribute from the associated policy/security attributes page (see table 20), then the command shall be terminated as described in 4.11.2.2.n ~~with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB~~.

If the BOOT EPOCH field contains zero or the boot epoch attribute in the Root Policy/Security attributes page (see 7.1.2.21) contains zero, then the contents of the BOOT EPOCH field shall be ignored. If the non-zero values in the BOOT EPOCH field and the boot epoch attribute in the Root Policy/Security attributes page do not match, then the command shall be terminated as described in 4.11.2.2.n ~~with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to INVALID FIELD IN CDB~~.

The ALLOWED PARTITION_ID field specifies the Partition_ID (see 4.6.4) of the partition to which access is allowed. If the ALLOWED PARTITION_ID field contains zero, the command shall be terminated as described in 4.11.2.2.n. ~~The command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB, if any of the following are true:~~

~~a) The ALLOWED PARTITION_ID field contains zero; or~~
~~b) The ALLOWED PARTITION_ID field contents do not match the contents of the PARTITION_ID field in the CDB.~~

The ALLOWED USER_OBJECT_ID field specifies the User_Object_ID (see 4.6.5) of the OSD object to which the capability allows access. If the ALLOWED USER_OBJECT_ID field contains zero and the command is not CREATE (see 6.4) or CREATE AND WRITE (see 6.5), then the command shall be terminated as described in 4.11.2.2.n. ~~The command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB, if any of the following are true:~~

~~a) The command is not CREATE (see 6.4) or CREATE AND WRITE (see 6.5), and the ALLOWED USER_OBJECT_ID field contains zero; or~~
~~a) The ALLOWED USER_OBJECT_ID field contents do not match the contents of the CDB USER_OBJECT_ID field or REQUESTED USER_OBJECT_ID field.~~

The ALLOWED RANGE LENGTH field specifies number of bytes in the range of user object bytes to which the capability allows access.

The ALLOWED RANGE STARTING BYTE OFFSET field specifies the location of the first byte in the range of user object bytes to which the capability allows access relative to the first byte (i.e., byte zero).

~~The command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB, if any of the following is true:~~

~~a) The range of bytes specified by the CDB LENGTH field and STARTING BYTE ADDRESS field in a CREATE AND WRITE command (see 6.5), READ command (see 6.23), or WRITE command (see 6.32) is not inside the range of bytes specified by the ALLOWED RANGE LENGTH field and ALLOWED RANGE STARTING BYTE OFFSET field;~~
~~b) The range of bytes specified by the CDB LENGTH field in an APPEND command (see 6.2) and the value in the user object logical length attribute in the User Object Information attributes page (see 7.1.2.11) is not inside the range of bytes specified by the ALLOWED RANGE LENGTH field and ALLOWED RANGE STARTING BYTE OFFSET field;~~
~~c) The range of bytes specified by the CDB CLEAR LENGTH field and CLEAR STARTING BYTE ADDRESS field in a CLEAR command (see 6.3) is not inside the range of bytes specified by the ALLOWED RANGE LENGTH field and ALLOWED RANGE STARTING BYTE OFFSET field;~~
~~d) The range of bytes specified by the CDB PUNCH LENGTH field and PUNCH STARTING BYTE ADDRESS field in a PUNCH command (see 6.20) is not inside the range of bytes specified by the ALLOWED RANGE LENGTH field and ALLOWED RANGE STARTING BYTE OFFSET field; or~~

~~e) The range of bytes from value in the CDB DATA MAP BYTE OFFSET field to the value in the user object logical length attribute in the User Object Information attributes page (see 7.1.2.11) in a READ MAP command (see 6.22) is not inside the range of bytes specified by the ALLOWED RANGE LENGTH field and ALLOWED RANGE STARTING BYTE OFFSET field.~~

If the ALLOWED RANGE LENGTH field is set to FFFF FFFF FFFF FFFFh and the ALLOWED RANGE STARTING BYTE OFFSET field is set to zero, then access is allowed to all bytes in the user object.

If the ALLOWED RANGE LENGTH field is set to FFFF FFFF FFFF FFFFh and the ALLOWED RANGE STARTING BYTE OFFSET field is set to a non-zero value, then access is allowed to all bytes from the allowed range starting byte to byte FFFF FFFF FFFF FFFFh. This shall not be considered an error.

The command that accesses a user object shall be terminated as described in 4.11.2.2.n, if none of the capabilities associated with the command (i.e., the capability in the CDB (see 5.2.1) and the capabilities, if any, in the CDB continuation segment (see 5.x)) match all of the following criteria:

a) The partition that contains the user object (e.g., the partition specified by the PARTITION_ID field in the CDB of a READ command) matches the ALLOWED PARTITION_ID field in the capability;
b) The user object being accessed (e.g., the user object specified by the USER_OBJECT_ID field in the CDB of a READ command) matches the ALLOWED USER_OBJECT_ID field in the capability; and
c) If data is allowed to be transferred to or from the user object (i.e., if the READ permission bit or the WRITE permission bit (see 4.11.2.2.1) is set to one), then the specified range of bytes being transferred (e.g., the range of bytes specified by the LENGTH field and STARTING BYTE ADDRESS field in the CDB of a READ command with the CDB CONTINUATION LENGTH field (see 5.2.1) is set to zero) is inside the range of bytes specified by the ALLOWED RANGE LENGTH field and ALLOWED RANGE STARTING BYTE OFFSET field.

{{The above validation requirement differs from those in OSD-2 r03 in two significant ways.

First, the contents of a single capability are required to match both the partition_id and the user_object_id of one user object being accessed. This is necessary to support, for example a COPY USER OBJECTS command in which two or more user objects are accessed.

Second, the exponential increase in the ways user objects can be accessed has obliged me use examples of what to test in place of a detailed list of all possible cases.}}

4.11.2.2.3 PAR capability object descriptor

If the object descriptor type is PAR (i.e., 2h), the OBJECT DESCRIPTOR field shall have the format shown in table 21, specifying a single partition to which the capability allows access. For a LIST COLLECTION command with the M_OBJECT bit set to one (see 4.11.2.2.1), the PAR capability object descriptor allows access to a single partition and the attributes associated with each collection in the partition. For the LIST command with the M_OBJECT bit set to one, the PAR capability object descriptor allows access to:

a) The root object and the attributes associated with each partition; or

b) A partition and the attributes associated with each user object in the partition.

Table 21 — Partition descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
60	(MSB)							
63				POLICY ACCESS TAG				(LSB)
64	(MSB)							
65				BOOT EPOCH				(LSB)
66								
71				Reserved				
72	(MSB)							
79				ALLOWED PARTITION_ID				(LSB)
80								
103				Reserved				

The POLICY ACCESS TAG field and BOOT EPOCH field are described in 4.11.2.2.2. {{Note: bug fix here.}}

The ALLOWED PARTITION_ID field specifies the partition to which access is allowed. The command shall be terminated as described in 4.11.2.2.n, if any of the following are true:

a) If the OBJECT TYPE field contains 02h (i.e., PARTITION), the command is not CREATE PARTITION (see 6.7), and the ALLOWED PARTITION_ID field contains zero; or
b) If the OBJECT TYPE field contains 01h (i.e., ROOT) and the ALLOWED PARTITION_ID field contains a value other than zero.

The command that accesses a partition shall be terminated as described in 4.11.2.2.n, if none of the capabilities associated with the command (i.e., the capability in the CDB (see 5.2.1) and the capabilities, if any, in the CDB continuation segment (see 5.x)) match all of the following criteria:

a) If the OBJECT TYPE field contains:
   A) 02h (i.e., PARTITION), then the partition being accessed (e.g., the partition specified by the PARTITION_ID field in the CDB of a LIST command) matches the ALLOWED PARTITION_ID field in the capability; or
   B) 01h (i.e., ROOT), then the partition being accessed (e.g., the partition specified by the PARTITION_ID field in the CDB of a LIST command) is zero;
   and
b) The User_Object_ID (see 4.6.2) associated with the object being accessed, if any, is zero.

~~The command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB, if any of the following are true:~~

~~a) The CDB USER_OBJECT_ID field, REQUESTED USER_OBJECT_ID field, COLLECTION_OBJECT_ID field or REQUESTED COLLECTION_OBJECT_ID field, if any, contains a value other than zero;~~
~~b) The OBJECT TYPE field contains 02h (i.e., PARTITION) and one of the following is true:~~
   ~~A) The command is not CREATE PARTITION (see 6.7) and the ALLOWED PARTITION_ID field contains zero; or~~

B) The ALLOWED PARTITION_ID field contents do not match the contents of the CDB PARTITION_ID field or REQUESTED PARTITION_ID field;

or

c) The OBJECT TYPE field contains 01h (i.e., ROOT) and one of the following is true:

A) The ALLOWED PARTITION_ID field contains a value other than zero; or

B) The CDB PARTITION_ID field, if any, contains a value other than zero.

4.11.2.2.4 COL capability object descriptor

If the object descriptor type is COL (i.e., 3h), the OBJECT DESCRIPTOR field shall have the format shown in table 22, specifying a single collection to which the capability allows access.If the M_OBJECT permission bit is set to one or the QUERY permission bit is set to one (see 4.11.2.2.1), the COL capability object descriptor allows access to a single collection and the attributes associated with each user object in the collection.

Table 22 — Collection descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
60	(MSB)			POLICY ACCESS TAG				
63								(LSB)
64	(MSB)			BOOT EPOCH				
65								(LSB)
66				Reserved				
71								
72	(MSB)			ALLOWED PARTITION_ID				
79								(LSB)
80	(MSB)			ALLOWED COLLECTION_OBJECT_ID				
87								(LSB)
88				Reserved				
103								

The POLICY ACCESS TAG field, BOOT EPOCH field, and ALLOWED PARTITION_ID field are described in 4.11.2.2.2. {{Bug fix here too!}}

The ALLOWED COLLECTION_OBJECT_ID field specifies the Collection_Object_ID (see 4.6.6) of the collection to which the capability allows access. If the ALLOWED COLLECTION_OBJECT_ID field contains zero and the command is not CREATE COLLECTION (see 6.6), then the command shall be terminated as described in 4.11.2.2.n. The command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB, if any of the following are true:

a) The command is not CREATE COLLECTION (see 6.6) and the ALLOWED COLLECTION_OBJECT_ID field contains zero; or

b) The ALLOWED COLLECTION_OBJECT_ID field contents do not match the contents of the CDB COLLECTION_OBJECT_ID field or REQUESTED COLLECTION_OBJECT_ID field.

The command that accesses a collection shall be terminated as described in 4.11.2.2.n, if none of the capabilities associated with the command (i.e., the capability in the CDB (see 5.2.1) and the capabilities, if any, in the CDB continuation segment (see 5.x)) match all of the following criteria:

a) The partition that contains the collection (e.g., the partition specified by the PARTITION_ID field in the CDB of a SET MEMBER ATTRIBUTES command) matches the ALLOWED PARTITION_ID field in the capability; and
b) The collection being accessed (e.g., the collection specified by the COLLECTION_OBJECT_ID field in the CDB of a SET MEMBER ATTRIBUTES command) matches the ALLOWED COLLECTION_OBJECT_ID field in the capability.

4.11.2.2.n Command termination due to errors detected in a capability

If an error is detected during the validation of a capability, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set as follows:

a) If the capability in which the error is detected is in the CDB (see 5.2.1), the additional sense code shall be set to INVALID FIELD IN CDB; or
b) If the capability in which the error is detected is in the CDB continuation segment (see 5.x), the additional sense code shall be set to INVALID FIELD IN PARAMETER LIST.

…

{{The subclause reordering restarts here to cover a small set of subclauses in which very minor changes are needed to support associating more than one capability with a command.}}

…

4.4 Elements of the example configuration

…

The policy/storage manager (see 4.11), if present, coordinates access constraints between OSD device servers and application clients, preparing the capabilities (see 3.1.4) application clients place in CDBs or CDB continuations segments (see 5.x) to gain access to OSD objects and command functions.

The security manager (see 4.12), if present, secures capabilities in cryptographically protected credentials (see 3.1.11) for OSD device servers and application clients.

…

4.11.3.2 Policy access tags

…

The device server terminates any command received with a capability (see 3.1.4) whose POLICY ACCESS TAG field contains a non-zero value that differs from the policy access tag attribute value in the Policy/Security attributes page associated with the object (see 4.11.2.2).

…

4.12.1 Basic security model

The OSD security model is a credential-based access control system composed of the following components:

a) An OBSD (see 3.1.27);
b) A policy/storage manager (see 4.11);
c) A security manager; and
d) Application clients.

The principal function of the security manager is preparing credentials (see 3.1.11) in response to application client requests. A credential is a data structure containing ~~a capability~~ one or more capabilities prepared by the policy/storage manager (see 4.11) and protected by ~~an integrity check value~~ one or two integrity check values (see 3.1.19), having the following properties:

a) The capability or capabilities (see 3.1.4) in the credential ~~grants~~ grant defined access to an OSD logical unit for specific command functions (see 3.1.10); and
b) The integrity check ~~value~~ values in the credential ~~protects~~ protect the ~~capability~~ capabilities and commands that include the ~~capability~~ capabilities from various attacks described in 4.12.4.

Figure 4 shows the flow of transactions between the components of the OSD security model.

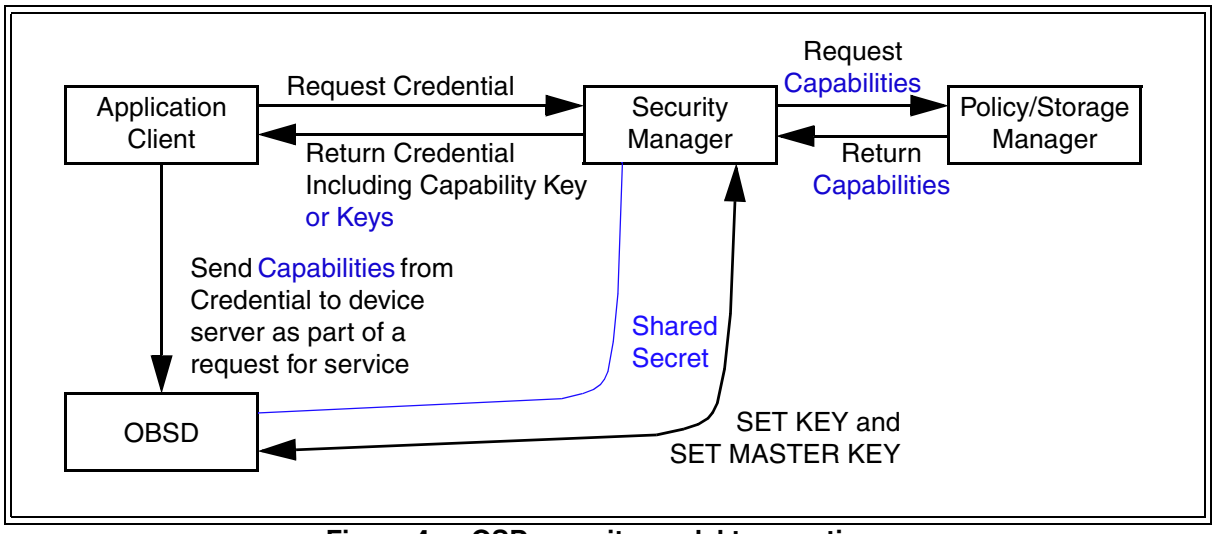

Figure 4 — OSD security model transactions

{{With the exception of 'Shared Secret', all the blue text in figure 4 is a replacement for existing text. To keep figure 4 manageable, strikeouts are not show for the old text.}}

The security manager generates credentials, including capabilities prepared by the policy/storage manager, for authorized application clients at the request of an application client. The security manager returns ~~a capability key~~ one or two capability keys with each credential. The credential gives the application client access to specific OSD components. The capability keys allow ~~key allows~~ the application client and device server to authenticate the commands and data they exchange with an integrity check value (see 4.12.8).

The protocol between the application client and the security manager is not defined by this standard. However, the structure of the credential returned from the security manager to the application client is.

If any security method except NOSEC is used, the device server validates each command received from an application client to confirm that:

a) The credential has not been tampered with (i.e., that the credential was generated by the security manager and includes an integrity check value using a secret key known only to the security manager and OSD device server); and
b) The credential was rightfully obtained by the application client from the security manager or through delegation by another application client (i.e., that the application client knows the ~~capability key that is~~ capability keys that are associated with the credential and has used the capability ~~key~~ keys to provide a proper integrity check value or values for the command); and
c) The requested command function is permitted by the capability or capabilities in the credential as described in 4.11.2.

The capability ~~key allows~~ keys allow the OSD device server to validate that an application client rightfully obtained a credential and that the capability or capabilities have ~~has~~ not been tampered with. An application client that has just ~~the~~ a capability (e.g., obtained by monitoring CDBs sent to the OSD device server) but not the associated capability key or keys is unable to generate commands with valid integrity check value, meaning that application client is denied access to the OSD logical unit. This protocol allows delegation of a credential if an application client delegates both the credential and the capability ~~key~~ keys.

The application client requests credentials and capability keys from the security manager for the command functions it needs to perform and sends those capabilities in those credentials to the OSD device server as part of commands that include an integrity check value using the capability ~~key~~ keys. While the application client is not trusted to follow this protocol, an application client that does not follow the protocol is unlikely to receive service from the OSD device server.

The security manager may authenticate the application client, but the OSD device server does not authenticate the application client. It is sufficient for the OSD device server to verify the capabilities and integrity check values sent by the application client.

…

4.12.7.2 Device server validation of request nonces

…

If the inputs to an integrity check value computation include a non-zero request nonce that is listed (see 4.12.7.3) as having been used in any previous integrity check value computation, the command shall be terminated with a CHECK CONDITION status, the sense key shall be set to ILLEGAL REQUEST, and the additional sense code shall be set to NONCE NOT UNIQUE. The command shall be terminated regardless of the success or failure of the previous command in which the duplicate request nonce appeared (e.g., the request nonce appearing in a WRITE command that ultimately fails due to insufficient quota or the request nonce appearing in a CREATE command that ultimately fails because the computed ~~credential~~ integrity check value for the credential is wrong shall not be accepted a second time).

…

4.12.9 Secret keys

4.12.9.1 Introduction

…

Table 33 — OSD secret key hierarchy

Key Name	Key Shared Using	Key Used To	Key Update Frequency
Keys shared between the security manager and the OSD device server			
Master	SET MASTER KEY command	Update Root key	Change of logical unit owner
Root	SET KEY command	Update Partition key	When Partition key may have been compromised (i.e., very infrequently)
Partition [a]	SET KEY command	Update Working keys	When Working key updates may have been compromised (i.e., infrequently)
Working [b]	SET KEY command	Create Capability keys	When normal key use affords too much chance that the working key might be reverse engineered (i.e., regularly)
Keys shared between the security manager and the application client [c]			
Capability [d]	Credentials and mechanisms not specified in this standard	Secure commands, responses, and data	New with each new Credential

[a] For the purposes of the secret key hierarchy, the root object is treated the same as any other partition OSD object using partition zero.

[b] For each partition, up to sixteen working keys may be active at any time, uniquely identified by the capability KEY VERSION field (see 4.11.2.2).

[c] The device server is capable of computing the capability key (see 4.12.6.3) using the reconstructed credential (see 4.12.6.2).

[d] As a dual purpose number, the capability key is different from other keys in the hierarchy. ~~The~~ A capability key is ~~the credential~~ one of the two integrity check values in a credential (see 4.12.5) ~~value~~. Even though the security manager computes it, the computation is based on values beyond the security manager's control (e.g., the user object to which the credential allows access). While changing the working key used to construct ~~the credential~~ integrity check values in a credential ~~value~~ invalidates the capability keys ~~key~~, ~~the credential~~ one or more of the capabilities in the credential may expire before that, making ~~the~~ those capability keys ~~key~~ invalid.

…

Change 4 – Increase integrity check value field sizes to allow HMAC-SHA-256 uses

Description

NIST has asked the security community to switch from SHA-1 to SHA-256 by 2010. These changes make that possible in OSD-2.

Proposed Changes in OSD-2 r03

{{Note: the Data-out integrity information format (see table 29) is covered as part of change 3.}}

…

Table 30 — Data-in integrity information format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			NUMBER OF COMMAND OR PARAMETER BYTES				
7								(LSB)
8	(MSB)			NUMBER OF RETRIEVED ATTRIBUTES BYTES				
15								(LSB)
16	(MSB)			DATA-IN INTEGRITY CHECK VALUE				
47~~35~~								(LSB)

…

4.12.8 Integrity check values

An integrity check value is a value produced by a cryptographic function (e.g., HMAC-SHA1) based on a secret key (see 4.12.9) that is able to be computed and verified by the entities knowing the secret key. Integrity check values are used to verify that:

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

Some integrity check value algorithms return values that contain fewer bytes than are available in the fields that this standard defines to contain integrity check values. If this occurs, then:

a) The integrity check value returned by the specified algorithm shall be placed in the field with the most significant byte of the integrity check value being placed in the most significant byte of the field and the remaining integrity check value bytes being placed in consecutive bytes of the field;
b) Zeros shall be placed in the unused bytes of the field up to and including the least significant byte of the field.

…

4.15.2.2 OSD response integrity check value sense data descriptor

The OSD response integrity check value sense data descriptor (see table 44) contains the response integrity check value used when the OSD security method is CMDRSP or ALLDATA (see 4.12.4).

Table 44 — OSD response integrity check value sense data descriptor format

Bit Byte	7	6	5	4	3	2	1	0
0	DESCRIPTOR TYPE (07h)							
1	ADDITIONAL LENGTH (20h ~~14h~~)							
2	(MSB)			RESPONSE INTEGRITY CHECK VALUE				
33~~21~~								(LSB)

…

5 Common Formats

5.1 OSD CDB format

…

Table 48 — Basic OSD CDB

Bit Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (7Fh)							
1	CONTROL							
2	Reserved							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved							
7	ADDITIONAL CDB LENGTH (~~192~~ 228)							
8	(MSB)			SERVICE ACTION				
9								(LSB)
10	Service action specific fields (see 5.2.1)							
235 ~~223~~								

…

5.2.1 Overview

…

Table 49 — OSD service action specific fields

Bit / Byte	7	6	5	4	3	2	1	0
10	Reserved			DPO [a]	FUA [a]	ISOLATION (see 5.2.5)		
… 183	… …							
184	Security parameters (see 5.2.8)							
235 ~~223~~								
…	…							

…

5.2.8 Security parameters

The CDB security parameters (see table 55) contain the security information needed for each command.

Table 55 — Security parameters format

Bit / Byte	7	6	5	4	3	2	1	0
183	Other CDB fields							
184	(MSB)	REQUEST INTEGRITY CHECK VALUE						
215								(LSB)
216	REQUEST NONCE							
227								
228	DATA-IN INTEGRITY CHECK VALUE OFFSET							
231								
232	DATA-OUT INTEGRITY CHECK VALUE OFFSET							
235								

{{The request integrity check value field is increased to 32 bytes. This increases the total size of the security parameters, which affects table 49, and all CDBs (also not shown except in CDB format tables included for other reasons.}}

The REQUEST INTEGRITY CHECK VALUE field contains an integrity check value (see 4.12.8) for the request sent by the application client. The REQUEST INTEGRITY CHECK VALUE field is used only by the CAPKEY security method, the CMDRSP security method, and the ALLDATA security method (see 4.12.4).

The CAPKEY security method for computing the request integrity check value is described in 4.12.4.3. The CMDRSP security method and ALLDATA security method for computing the request integrity check value is described in 4.12.4.4.

The device server shall validate the request integrity check value as described in 4.12.6.1.

…

7.1.2.29 Current Command attributes page

The Current Command attributes page (FFFF FFFEh) shall contain the attributes listed in table 180.

Table 180 — Current Command attributes page contents

Attribute Number	Length (bytes)	Attribute	Application Client Settable	OSD Logical Unit Provided
0h	40	Page identification	No	Yes
1h	32 ~~20~~	Response integrity check value	No	Yes
2h	1	Object Type	No	Yes
3h	8	Partition_ID	No	Yes
4h	8	Collection_Object_ID or User_Object_ID	No	Yes
5h	8	Starting byte address of append	No	Yes
6h to FFFF FFFEh		Reserved	No	

…

The page format for the Current Command attributes page is shown in table 181.

Table 181 — Current Command attributes page format

Bit Byte	7	6	5	4	3	2	1	0
0	(MSB)							
3				PAGE NUMBER (FFFF FFFEh)				(LSB)
4	(MSB)							
7				PAGE LENGTH (3Ch ~~30h~~)				(LSB)
8	(MSB)							
39~~27~~				RESPONSE INTEGRITY CHECK VALUE				(LSB)
40~~28~~				OBJECT TYPE				
41~~29~~								
43~~31~~				Reserved				
44~~32~~	(MSB)							
51~~39~~				PARTITION_ID				(LSB)
52~~40~~	(MSB)							
59~~47~~				COLLECTION_OBJECT_ID OR USER_OBJECT_ID				(LSB)
60~~48~~	(MSB)							
67~~55~~				STARTING BYTE ADDRESS OF APPEND				(LSB)

…

Change 5 – Scatter/Gather List additions

Description

This change defines a CDB continuation descriptor to allow scatter/gather transfers in CREATE AND WRITE commands, READ commands, and WRITE commands. The APPEND command is not proposed to have scatter/gather capability due to the difficulty of defining stable offset values for the scatter/gather list. When scatter/gather processing is desired, the WRITE command should be used instead of the APPEND command.

This change does not require the placement of capabilities in the CDB continuation segment.

Proposed Changes in OSD-2 r03

5.y.1 Overview

…

The CDB CONTINUATION DESCRIPTOR TYPE field (see table x4) specifies the format of the CDB continuation descriptor type specific data.

Table x4 — CDB CONTINUATION DESCRIPTOR TYPE field

Value	Description	Reference
0000h	No more continuation descriptors [a]	
0001h	Scatter/gather list	5.y.c
FFEEh	Extension capabilities	5.y.z
{{Other rows of this table are filled in by other changes in this proposal.}} {{The most complete body for this table can be found in change 7 table x4.}}		
all other values	Reserved	

[a] Since the CDB continuation segment pad bytes, if any, are set to zero (see 5.x), encountering a CDB continuation descriptor type of zero shall be processed in the same way as reaching the last byte of the CDB continuation segment.

…

5.y.c Scatter/gather list

{{All of 5.y.c is new. The use of change markups is suspended for the remainder of 5.y.c.}}

The scatter/gather list CDB continuation descriptor (see table x6) specifies the relationship between contents of the command data buffer segment (see 4.14.3 for the Data-In Buffer or 4.14.4 for the Data-Out Buffer) and the bytes in the user object.

Table x6 — Scatter/gather list CDB continuation descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
	CDB continuation descriptor header							
0	(MSB)			CDB CONTINUATION DESCRIPTOR TYPE (0001h)				
1								(LSB)
2	Reserved							
3	Reserved					PAD LENGTH (000b)		
4	(MSB)			CONTINUATION DESCRIPTOR LENGTH (n-7)				
7								(LSB)
	CDB continuation descriptor type specific data							
8	Scatter/gather list entry [first] (see table x7)							
23								
	⋮							
n-15	Scatter/gather list entry [last] (see table x7)							
n								

The CDB CONTINUATION DESCRIPTOR TYPE field contains 0001h (i.e., scatter/gather list CDB continuation descriptor).

The PAD LENGTH field is set to zero to indicate that no pad bytes are needed to eight byte align a scatter/gather list CDB continuation descriptor. If the PAD LENGTH field is not set to zero in a scatter/gather list CDB continuation descriptor, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The CONTINUATION DESCRIPTOR LENGTH field specifies the number of bytes that follow in this descriptor.

Each scatter/gather list entry (see table x7) specifies the starting byte offset in the user object and number of bytes to be transferred to or from the command data buffer segment.

The first byte in the command data buffer segment is transferred to or from the user object byte offset indicated by the first scatter/gather list entry. Additional bytes are transferred to or from the command data buffer segment byte by byte until the number of bytes indicated by the first scatter/gather list entry have been transferred.

The next byte in the command data buffer segment (i.e., the first byte in the command data buffer segment that was not transferred by the first scatter/gather list entry) is transferred to the user object byte offset indicated by the second scatter/gather list entry. Additional bytes are transferred to or from the command data buffer segment byte by byte until the number of bytes indicated by the second scatter/gather list entry have been transferred.

This process is repeated until all the bytes indicated by the CDB LENGTH field have been transferred or all the scatter/gather list entries have been processed, which ever occurs first.

Each scatter/gather list entry has the format shown in table x7.

Table x7 — Scatter/gather list entry format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
7				USER OBJECT BYTE OFFSET				(LSB)
8	(MSB)							
15				BYTES TO TRANSFER				(LSB)

The USER OBJECT BYTE OFFSET field specifies the starting byte offset in the user object for this scatter/gather list entry.

The BYTES TO TRANSFER field specifies the number of bytes to transfer for this scatter/gather list entry.

It shall not be an error for the byte ranges specified by individual scatter/gather list entries to overlap (e.g., the second scatter/gather list entry may transfer some or all of the same bytes that were transferred by the first scatter/gather list entry).

If the values in the BYTES TO TRANSFER field and USER OBJECT BYTE OFFSET field result an attempt to read a byte that is beyond the user object logical length attribute value in the User Object Information attributes page (see 7.1.2.11), then:

a) The bytes between the user object byte offset and the user object logical length shall be transferred;
b) The command shall be terminated with CHECK CONDITION status, with the sense key shall be set to RECOVERED ERROR and the additional sense code set to READ PAST END OF USER OBJECT;
c) The command-specific information sense data descriptor (see SPC-3) shall be included in the sense data; and
d) The COMMAND-SPECIFIC INFORMATION field shall contain the number of bytes transferred by the command, including but not limited to the bytes transferred by this scatter/gather list entry.

…

6.5 CREATE AND WRITE

{{Add the CDB CONTINUATION LENGTH field to bytes 48 to 51 of table 61 as shown in change 2.}}

…

The contents of the STARTING BYTE ADDRESS field are defined in 5.2.9. If the CDB continuation segment (see 5.x), if any, contains a scatter/gather list CDB continuation descriptor and the STARTING BYTE ADDRESS field contains a value other than zero, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

…

The contents of the CDB CONTINUATION LENGTH field are defined in 5.2.x. If the CDB CONTINUATION LENGTH field is not set to zero and the CDB continuation segment (see 5.x) contains a scatter/gather list CDB continuation descriptor, that descriptor shall be processed as described in 5.y.c.

The command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST, if CDB continuation segment (see 5.x):

a) The CDB continuation segment contains more than one scatter/gather list CDB continuation descriptor; or
b) The CDB continuation segment contains any CDB continuation descriptors other than the scatter/gather list CDB continuation descriptor.

The get and set attributes parameters are defined in 5.2.4. …

…

6.23 READ

{{Add the CDB CONTINUATION LENGTH field to bytes 48 to 51 of table 107 as shown in change 2.}}

…

The contents of the STARTING BYTE ADDRESS field are defined in 5.2.9.

If the STARTING BYTE ADDRESS field specifies a byte that is beyond the user object logical length attribute value in the User Object Information attributes page (see 7.1.2.11), then:

a) No bytes shall be transferred; and
b) The command shall be terminated with ~~a~~ CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

If the CDB continuation segment (see 5.x), if any, contains a scatter/gather list CDB continuation descriptor and the STARTING BYTE ADDRESS field contains a value other than zero, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

…

The contents of the CDB CONTINUATION LENGTH field are defined in 5.2.x. If the CDB CONTINUATION LENGTH field is not set to zero and the CDB continuation segment (see 5.x) contains a scatter/gather list CDB continuation descriptor, that descriptor shall be processed as described in 5.y.c.

The command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST, if CDB continuation segment (see 5.x):

a) The CDB continuation segment contains more than one scatter/gather list CDB continuation descriptor; or
b) The CDB continuation segment contains any CDB continuation descriptors other than the scatter/gather list CDB continuation descriptor.

The get and set attributes parameters are defined in 5.2.4. …

…

6.32 WRITE

{{Add the CDB CONTINUATION LENGTH field to bytes 48 to 51 of table 120 as shown in change 2.}}

…

The contents of the STARTING BYTE ADDRESS field are defined in 5.2.9. If the CDB continuation segment (see 5.x), if any, contains a scatter/gather list CDB continuation descriptor and the STARTING BYTE ADDRESS field contains a value other than zero, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST, and the additional sense code set to INVALID FIELD IN CDB.

…

The contents of the CDB CONTINUATION LENGTH field are defined in 5.2.x. If the CDB CONTINUATION LENGTH field is not set to zero and the CDB continuation segment (see 5.x) contains a scatter/gather list CDB continuation descriptor, that descriptor shall be processed as described in 5.y.c.

The command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST, if CDB continuation segment (see 5.x):

a) The CDB continuation segment contains more than one scatter/gather list CDB continuation descriptor; or
b) The CDB continuation segment contains any CDB continuation descriptors other than the scatter/gather list CDB continuation descriptor.

The get and set attributes parameters are defined in 5.2.4. …

Change 6 – Object duplication model

Description

The definition of the COPY USER OBJECTS command (see change 7) requires that portions of the snapshot/clone model be introduced by this proposal.

Proposed Changes in OSD-2 r03

4.d Object duplication

{{All of 4.d is new. The use of change markups is suspended for the remainder of 4.d. It is suggested that 4.d be placed between 4.11 (Policy/Storage management) and 4.12 (Security).}}

4.d.1 Overview

The following mechanisms are defined for duplicating the data and attributes contained in one or more user objects and collections in new user objects and collections:

a) The CREATE SNAPSHOT command (see 4.d.2);
b) The CREATE CLONE command (see 4.d.2); and
c) The COPY USER OBJECTS command (see 6.h).

A model for the partition snapshot and clone mechanisms appears in 4.d.2.

The COPY USER OBJECTS command:

1) Creates a destination user object; and
2) Copies the data and maybe the attributes from one or more source user objects to that destination user object using:
   A) The object duplication methods described in 4.d.3;
   B) The object duplication state management methods described in 4.d.4; and
   C) The object duplication space accounting methods described in 4.d.5.

4.d.2 Partition snapshots and clones

{{See 08-182.}}

4.d.3 Object duplication methods

Duplicating user object data, collection data, partition data, and attributes may or may not involve making two copies of the same bytes on stable storage (see table x8).

Table x8 — Object duplication methods

Name	Code [a]	Description
DEFAULT	00h	Used to specify one of the other codes in this table that is selected via a specified attribute value.
SPACE EFFICIENT	01h	Duplicated bytes belong to a particular object only when specific application client actions necessitate it (e.g., a copy-on-write mechanism in which duplicated bytes become associated with a particular object only when changes in their contents necessitate it).
PRE-ALLOCATED COPY ON WRITE	41h	Similar to SPACE EFFICIENT except that bytes are reserved for physical copies of all duplicated bytes (e.g., the reserved data space attribute in the User Object Information attributes page (see 7.1.2.11) is set to ensure that space is available for all duplicated bytes when application client actions necessitate copying them).
BYTE BY BYTE COPY	81h	A copy shall be made of every duplicated byte, and each object shall have its own, unique copy of the duplicated bytes.
FASTER COPY PERFORMANCE	FDh	A vendor specific object duplication mechanism whose characteristics are similar to those of the SPACE EFFICIENT object duplication method.
HIGHER DATA DUPLICATON	FEh	A vendor specific object duplication mechanism whose characteristics are similar to those of the BYTE BY BYTE COPY object duplication method.
DO NOT CARE	FFh	The device sever may use any duplication method or combination of methods.

[a] These codes are used in fields, attribute numbers, and attribute values. All codes not listed in this table are reserved.

Except for the BYTE BY BYTE COPY object duplication method, the device server may use the object duplication method specified by the application client as a recommendation while still employing aspects of any supported object duplication method to achieve optimum processing times and/or space utilization.

Some object duplication methods (e.g., the SPACE EFFICIENT) cause single instances of recorded data to be shared among multiple objects. This type of shared data shall not be removed from stable storage until there are no objects that reference it.

Support for the various object duplication methods is indicated by attributes in the Root Information attributes page (see 7.1.2.8).

4.d.4 Object duplication state management

Duplicating user object data, collection data, partition data, and attributes may take a significant interval of time. The following mechanisms are provided so that application clients may specify how changes in source objects are to be handled during a duplication operation:

a) Time of duplication (see 4.d.4.1); and
b) Source object freeze (see 4.d.4.2).

These mechanisms are complementary. The use of one tends to eliminate the need to use any of the others.

4.d.4.1 Time of duplication source object management

Time of duplication codes (see table x9) allow the application client to specify what time in the life of a source object is to be used for purposes of duplicating that object. If multiple objects are duplicated by a single command, the time of duplication requirements apply separately to each duplicated object.

Table x9 — Time of duplication source object management

Name	Code [a]	Description
DEFAULT	0h	Used to specify one of the other codes in this table that is selected via a specified attribute value.
BEGINNING	1h	The duplicated object shall have the contents of the source object at the time the duplication was begun.
DO NOT CARE	8h	The duplicated object may have any contents of the source object, including contents that were not in effect at either the beginning or the end of the duplication.
END	Fh	The duplicated object shall have the contents of the source object at the time the duplication was completed.

[a] These codes are used in field, attribute numbers, and attribute values. All codes not listed in this table are reserved.

Support for the various time of duplication methods is indicated by attributes in the Root Information attributes page (see 7.1.2.8).

4.d.4.2 Source object freeze duplication management

A way to ensure the state of a source object during duplication is to set the object's object accessibility attribute (e.g., the object accessibility attribute in the User Object Information attributes page (see 7.1.2.11)) to disable write access (i.e., to 0000 00001h). This is described as freezing the source object.

Support for source object freeze duplication management is indicated by the support for duplicated object freezing attribute in the Root Information attributes page (see 7.1.2.8).

4.d.5 Object duplication space accounting

Some object duplication methods (e.g., the SPACE EFFICIENT object duplication method described in table x8 (see 4.d.3)) result in a situation where the used capacity (e.g., the used capacity attribute in a Partition Information attributes page (see 7.1.2.9)) of the original object plus the used capacity of the duplicate object almost equals the used capacity of the original object alone. However, such situations also may evolve in ways where the used capacity increases for reasons that are not obvious consequences of the commands being processed. Such increases in the used capacity attribute value shall not result in CHECK CONDITION status being returned for a command that has already begun processing, but they result in quota errors being generated (see 4.10.2) for future commands.

To assist application clients in managing capacity usage and the quotas (see 4.10) on capacity usage, the potential used capacity increment attribute in the Partition Information attributes page (see 7.1.2.9) indicates the maximum number of bytes by which the used capacity attribute in the same Partition Information attributes page might increase due to ongoing command processing.

{{No differences could be detected between the attribute called charged-to-partition in the Snapshots proposal v3.14 and the attribute called used capacity in OSD-2 r03. Therefore, no charged-to-partition attribute has been created.}}

…

7.1.2.8 Root Information attributes page

The Root Information attributes page (R+1h) shall contain the attributes listed in table 127.

Table 127 — Root Information attributes page contents

Attribute Number	Length (bytes)	Attribute	Application Client Settable	OSD Logical Unit Provided
…	…	…	…	…
123h	1	Data/attributes atomicity multiplier	No	Yes
124h to 1FFh		Reserved	No	
200h to 2FFh	0 or 4	Supported object duplication method	No	Yes
300h to 30Fh	0 or 4	Supported time of duplication method	No	Yes
310h	0 or 4	Support for duplicated object freezing	No	Yes
~~124h~~ 311h to FFFF FFFEh		Reserved	No	
…	…	…	…	…

…

The used capacity attribute (number 81h) shall contain the number of bytes used by all root object attributes, partitions, collections and user objects stored by the OSD logical unit including attributes bytes for the partition, collections, and user objects. If any objects in the OSD logical unit are the result of object duplications (see 4.d), the value of the used capacity attribute may increase for reasons that are not obvious consequences of the commands being processed as described in 4.d.5.

…

Each supported object duplication method attribute (numbers 200h to 2FFh) shall contain support information for one object duplication method. The attribute number is 200h plus the code shown in table x8 (see 4.d.3) for the object duplication method for which support information is being provided. If an attribute in the range 200h to 2FFh is undefined (see 3.1.50), then the associated object duplication method is not supported for any usage. If an attribute in the range 200h to 2FFh is defined (see 3.1.15), then the attribute value (see table x10) indicates the functions for which the object duplication method is supported.

Table x10 — Supported object duplication method attributes contents

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved							SNAPSHOT
1	Reserved							CLONE
2	Reserved							
3	Reserved							COPY_UO

…

If the SNAPSHOT bit is set to zero, the object duplication method indicated by the attribute number is not supported for use by the CREATE SNAPSHOT command (see TBD {{in 08-182}}). If the SNAPSHOT bit is set to one, the object duplication method indicated by the attribute number is supported for use by the CREATE SNAPSHOT command.

If the CLONE bit is set to zero, the object duplication method indicated by the attribute number is not supported for use by the CREATE CLONE command (see TBD {{in 08-182}}). If the CLONE bit is set to one, the object duplication method indicated by the attribute number is supported for use by the CREATE CLONE command.

If the COPY_UO bit is set to zero, the object duplication method indicated by the attribute number is not supported for use by the COPY USER OBJECTS command (see 6.h). If the COPY_UO bit is set to one, the object duplication method indicated by the attribute number is supported for use by the COPY USER OBJECTS command.

If any form of object duplication is supported (see 4.d), attribute number 200h (i.e., the supported object duplication method attribute for the DEFAULT object duplication method) and attribute number 2FFh (i.e., the supported object duplication method attribute for the DO NOTE CARE object duplication method) shall be defined (see 3.1.15) and the attribute value shall be FFFF FFFFh (i.e., all uses of the DEFAULT object duplication method and the DO NOT CARE object duplication method shall be supported). The value of attribute number 200h or attribute number 2FFh should not be used to determine which object duplication commands are supported. This information is returned by the REPORT SUPPORTED OPERATION CODES command (see 6.18 and SPC-4).

Each supported time of duplication method attribute (numbers 300h to 30Fh) shall contain support information for one time of duplication source object management method. The attribute number is 300h plus the code shown in table x9 (see 4.d.4.1) for the time of duplication method for which support information is being provided. If an attribute in the range 300h to 30Fh is undefined (see 3.1.50), then the associated time of duplication method is not supported for any usage. If an attribute in the range 300h to 30Fh is defined (see 3.1.15), then the attribute value (see table x11) indicates the functions for which the object duplication method is supported.

Table x11 — Supported time of duplication method attributes contents

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved							SNAPSHOT
1	Reserved							CLONE
2	Reserved							
3	Reserved							COPY_UO

…

If the SNAPSHOT bit is set to zero, the time of duplication method indicated by the attribute number is not supported for use by the CREATE SNAPSHOT command (see TBD {{in 08-182}}). If the SNAPSHOT bit is set to one, the time of duplication method indicated by the attribute number is supported for use by the CREATE SNAPSHOT command.

If the CLONE bit is set to zero, the time of duplication method indicated by the attribute number is not supported for use by the CREATE CLONE command (see TBD {{in 08-182}}). If the CLONE bit is set to one, the time of duplication method indicated by the attribute number is supported for use by the CREATE CLONE command.

If the COPY_UO bit is set to zero, the time of duplication method indicated by the attribute number is not supported for use by the COPY USER OBJECTS command (see 6.h). If the COPY_UO bit is set to one, the time of duplication method indicated by the attribute number is supported for use by the COPY USER OBJECTS command.

If any form of time of duplication source object management is supported (see 4.d.4.1), attribute number 300h (i.e., the supported time of duplication method attribute for the DEFAULT time of duplication method) and attribute number 308h (i.e., the supported time of duplication method attribute for the DO NOT CARE time of duplication method) shall be defined (see 3.1.15) and the attribute value shall be FFFF FFFFh (i.e., all uses of the DEFAULT time of duplication method and the DO NOT CARE time of duplication method) shall be supported if any time of duplication source object management is supported).

The support for duplicated object freezing attribute (number 310h) shall contain support information for source object freeze duplication management (see 4.d.4.2). If the support for duplicated object freezing attribute is undefined (see 3.1.50), then source object freeze duplication management is not supported for any usage. If the support for duplicated object freezing attribute is defined (see 3.1.15), then the attribute value (see table x12) indicates the functions for which source object freeze duplication management is supported.

Table x12 — Support for duplicated object freezing attribute contents

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved							SNAPSHOT
1	Reserved							CLONE
2	Reserved							
3	Reserved							COPY_UO

…

If the SNAPSHOT bit is set to zero, source object freeze duplication management (see 4.d.4.2) is not supported for use by the CREATE SNAPSHOT command (see TBD {{in 08-182}}). If the SNAPSHOT bit is set to one, source object freeze duplication management is supported for use by the CREATE SNAPSHOT command.

If the CLONE bit is set to zero, source object freeze duplication management (see 4.d.4.2) is not supported for use by the CREATE CLONE command (see TBD {{in 08-182}}). If the CLONE bit is set to one, source object freeze duplication management is supported for use by the CREATE CLONE command.

If the COPY_UO bit is set to zero, source object freeze duplication management (see 4.d.4.2) is not supported for use by the COPY USER OBJECTS command (see 6.h). If the COPY_UO bit is set to one, source object freeze duplication management is supported for use by the COPY USER OBJECTS command.

{{The supported object duplication method attributes, supported time of duplication method attributes, and support for duplicated object freezing attribute (no s) must be added to Annex B too.}}

7.1.2.9 Partition Information attributes page

The Partition Information attributes page (P+1h) shall contain the attributes listed in table 131.

Table 131 — Partition Information attributes page contents

Attribute Number	Length (bytes)	Attribute	Application Client Settable	OSD Logical Unit Provided
…	…	…	…	…
83h	4	Object accessibility	Yes	No
84h	0 or 8	Potential used capacity increment	No	Yes
~~84h~~ 85h to BFh		Reserved	No	
…	…	…	…	…
D2h	0 or 8	Reserved data space	Yes	No
D3h to 1FFh		Reserved	No	
200h	0 or 4	Default snapshot duplication method	Yes	No
201h	0 or 4	Default clone duplication method	Yes	No
202h	0 or 4	Default copy user objects duplication method	Yes	No
203 to 1FFh		Reserved	No	
300h	0 or 4	Default snapshot time of duplication method	Yes	No
301h	0 or 4	Default clone time of duplication method	Yes	No
302h	0 or 4	Default copy user objects time of duplication method	Yes	No
~~D3h~~ 303h to FFFF FFFEh		Reserved	No	
…	…	…	…	…

…

For all partitions except partition zero, the used capacity attribute (number 81h) shall contain the number of allocated bytes for the partition as described in this subclause. For partition zero, the used capacity attribute shall contain the number of allocated bytes for partition zero and all other partitions described in this subclause. The number of allocated bytes shall be computed as the sum of the following:

a) The number of bytes used by:
   A) The partition or partitions;
   B) All collections within the partition or partitions; and
   C) All user objects within the partition or partitions including attributes bytes; ~~and~~ and
b) The number of unused reserved bytes computed as:
   A) Value in the reserved data space attribute in this Partition Information attributes page minus the value in the actual data space attribute in this Partition Information attributes page; or

B) Zero if the value in the actual data space attribute in this Partition Information attributes page is larger than the value in the reserved data space attribute in this Partition Information attributes page.

If any object in the partition the result of object duplications (see 4.d), the value of the used capacity attribute may increase for reasons that are not obvious consequences of the commands being processed as described in 4.d.5.

…

The potential used capacity increment (number 84h) shall contain the maximum number of bytes by which the used capacity attribute in this Partition Information attributes page might increase due to ongoing command processing as described in 4.d.5.

…

If snapshot object duplication is supported (see 4.d), the default snapshot duplication method attribute (number 200h) shall be defined (see 3.1.15) and shall contain one of the codes in table x8 (see 4.d.3) other than DEFAULT. A CREATE PARTITION command (see 6.6) shall set the default snapshot duplication method attribute to DO NOT CARE (see table x8). If set attributes list (see 5.2.4.4) contains an entry attempts to set the default snapshot duplication method attribute to DEFAULT or to a code that the supported object duplication method attributes in the Root Information attributes page (see 7.1.2.8) indicate is not supported, then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. {{This text will need to be revised to be consistent with 08-181.}}

If clone object duplication is supported (see 4.d), the default clone duplication method attribute (number 201h) shall be defined (see 3.1.15) and shall contain one of the codes in table x8 (see 4.d.3) other than DEFAULT. A CREATE PARTITION command (see 6.6) shall set the default clone duplication method attribute to DO NOT CARE (see table x8). If set attributes list (see 5.2.4.4) contains an entry attempts to set the default clone duplication method attribute to DEFAULT or to a code that the supported object duplication method attributes in the Root Information attributes page (see 7.1.2.8) indicate is not supported, then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. {{This text will need to be revised to be consistent with 08-181.}}

If the copy user objects form of object duplication is supported (see 4.d), the default copy user objects duplication method attribute (number 202h) shall be defined (see 3.1.15) and shall contain one of the codes in table x8 (see 4.d.3) other than DEFAULT. A CREATE PARTITION command (see 6.6) shall set the default copy user objects duplication method attribute to DO NOT CARE (see table x8). If set attributes list (see 5.2.4.4) contains an entry attempts to set the default copy user objects duplication method attribute to DEFAULT or to a code that the supported object duplication method attributes in the Root Information attributes page (see 7.1.2.8) indicate is not supported, then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. {{This text will need to be revised to be consistent with 08-181.}}

If snapshot object duplication is supported (see 4.d), the default snapshot time of duplication method attribute (number 300h) shall be defined (see 3.1.15) and shall contain one of the codes in table x9 (see 4.d.4.1) other than DEFAULT. A CREATE PARTITION command (see 6.6) shall set the default snapshot time of duplication method attribute to DO NOT CARE (see table x9). If set attributes list (see 5.2.4.4) contains an entry attempts to set the default snapshot time of duplication method attribute to DEFAULT or to a code that the supported time of duplication method attributes in the Root Information attributes page (see 7.1.2.8) indicate is not supported, then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. {{This text will need to be revised to be consistent with 08-181.}}

If clone object duplication is supported (see 4.d), the default clone time of duplication method attribute (number 301h) shall be defined (see 3.1.15) and shall contain one of the codes in table x9 (see 4.d.4.1) other than

DEFAULT. A CREATE PARTITION command (see 6.6) shall set the default clone time of duplication method attribute to DO NOT CARE (see table x9). If set attributes list (see 5.2.4.4) contains an entry attempts to set the default clone time of duplication method attribute to DEFAULT or to a code that the supported time of duplication method attributes in the Root Information attributes page (see 7.1.2.8) indicate is not supported, then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. {{This text will need to be revised to be consistent with 08-181.}}

If the copy user objects form of object duplication is supported (see 4.d), the default copy user objects time of duplication method attribute (number 303h) shall be defined (see 3.1.15) and shall contain one of the codes in table x9 (see 4.d.4.1) other than DEFAULT. A CREATE PARTITION command (see 6.6) shall set the default copy user objects time of duplication method attribute to DO NOT CARE (see table x9). If set attributes list (see 5.2.4.4) contains an entry attempts to set the default copy user objects time of duplication method attribute to DEFAULT or to a code that the supported time of duplication method attributes in the Root Information attributes page (see 7.1.2.8) indicate is not supported, then the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST. {{This text will need to be revised to be consistent with 08-181.}}

{{The potential used capacity increment attribute, default snapshot duplication method attribute, default clone duplication method attribute, default copy user objects duplication method attribute, default snapshot time of duplication method attribute, default clone time of duplication method attribute, and default copy user objects time of duplication method attribute must be added to Annex B too.}}

…

7.1.2.10 Collection Information attributes page

…

The used capacity attribute (number 81h) shall contain the number of bytes used by the collection including attributes bytes. If the collection the result of an object duplication (see 4.d), the value of the used capacity attribute may increase for reasons that are not obvious consequences of the commands being processed as described in 4.d.5.

…

7.1.2.11 User Object Information attributes page

…

The used capacity attribute (number 81h) shall contain the sum of:

a) The number of bytes used by the user object including attributes bytes; and
b) The number of unused reserved bytes computed as:
   A) Value in the reserved data space attribute in this User Object Information attributes page minus the value in the actual data space attribute in this User Object Information attributes page; or
   B) Zero if the value in the actual data space attribute in this User Object Information attributes page is larger than the value in the reserved data space attribute in this User Object Information attributes page.

If the user object the result of an object duplication (see 4.d), the value of the used capacity attribute may increase for reasons that are not obvious consequences of the commands being processed as described in 4.d.5.

…

Change 7 – Definition of a COPY USER OBJECTS command

Description

This change defines an approximation of the object_copy function described in a half page of the OSD-Snapshot-proposal-v3.14.

The CDB will not accommodate a new capability object descriptor format that contains two complete user object descriptions (policy access tag, partition/user object IDs, byte ranges – an additional 36 bytes) plus a second user object byte range (partition/user object ID, byte range – an additional 32 bytes), so this change places the destination object information in the CDB and the source object information in the CDB continuation segment.

This proposal assumes that the destination user object does not already exist and is to be created by the command (a la CREATE AND WRITE). The traditions of the OSD standard dictate that a command cannot optionally create a user object if it does not already exist (e.g., the CREATE AND WRITE command verses the WRITE command). If the intention of the OSD-Snapshot-proposal-v3.14 object_copy function was that the user object already existed prior to the copy command processing, this proposal will need to be modified accordingly.

This proposal extends the OSD-Snapshot-proposal-v3.14 object_copy function in the following ways:

- Multiple byte ranges are allowed in a single copy command.
- Copies from multiple user objects to a single destination object are allowed. (The DOS copy command has provided this function since the age of stone knives and bear skins, why not OSD-2?)

This change makes use of all the CDB continuation features defined elsewhere in this proposal.

Proposed Changes in OSD-2 r03

Table 23 — Commands allowed by specific capability field values

Commands allowed and CDB fields whose contents are restricted by capability field contents, if any	Capability Field values that allow a command		
	Object Type Name	Permission Bits That Are Set To One	Object Descriptor Name
…	…	…	…
A COPY USER OBJECTS command with one destination user object and one or more source user objects [a]			
Destination user object	USER	CREATE and WRITE	USER
Source user object or user objects	USER	READ	USER
…	…	…	…
Combinations of OBJECT TYPE field, PERMISSION BITS field, and OBJECT DESCRIPTOR TYPE field values not shown in this table and table 24 are reserved. The capability fields not shown in this table may place additional limits on the objects that are allowed to be accessed.			
[a] This command accesses multiple objects. One capability is necessary for each object accessed. The solo capability (see 3.1.b) appears in the CDB (see 5.2.1). The other capabilities appear in the CDB continuation segment (see 5.x).			

…

Table 47 — OSD commands that are allowed in the presence of various reservations

OSD Command	Addressed logical unit has this type of persistent reservation held by another I_T nexus				
	From any I_T nexus		From registered I_T nexus (RR all types)	From not registered I_T nexus	
	Write Excl	Excl Access		Write Excl RR	Excl Access – RR
…	…	…	…	…	…
COPY USER OBJECTS	Conflict	Conflict	Allowed	Conflict	Conflict
…	…	…	…	…	…
Key: **Excl**=Exclusive, **RR**=Registrants Only or All Registrants					

…

Table 57 — Commands for OSD type devices

Command name	Operation code	Service action [a]	Type	Reference
…	…	…	…	…
COPY USER OBJECTS	7Fh	8893h	M	6.h

Type Key: M = Command implementation is mandatory.
O = Command implementation is optional.

[a] No entry in the service action column means that the SERVICE ACTION field does not apply to the command. Service action codes values between 8800h and 8F7Fh that are not listed in this table are reserved for future standardization. Service action code values between 8F80h and 8FFFh may have vendor specific command assignments.

[b] …

…

5.y.1 Overview

…

The CDB CONTINUATION DESCRIPTOR TYPE field (see table x4) specifies the format of the CDB continuation descriptor type specific data.

Table x4 — CDB CONTINUATION DESCRIPTOR TYPE field

Value	Description	Reference
0000h	No more continuation descriptors [a]	
0001h	Scatter/gather list	5.y.c
0101h	Copy user object source	5.y.h
FFEEh	Extension capabilities	5.y.z
{{This is the most complete version of table x4. However, when copying the table, the first instance of table x4 should be used for the table title and basic format. The table rows can be copied from this table after the blue-line formatting is removed.}}		
all other values	Reserved	

[a] Since the CDB continuation segment pad bytes, if any, are set to zero (see 5.x), encountering a CDB continuation descriptor type of zero shall be processed in the same way as reaching the last byte of the CDB continuation segment.

…

5.y.h Copy user object source

{{All of 5.y.h is new. The use of change markups is suspended for the remainder of 5.y.h.}}

The copy user object source CDB continuation descriptor (see table x13) specifies all or part of a user object as the source of bytes for a command (e.g., the source of bytes for a COPY USER OBJECTS command (see 6.h)).

Table x13 — Copy user object source CDB continuation descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
	CDB continuation descriptor header							
0	(MSB)			CDB CONTINUATION DESCRIPTOR TYPE (0101h)				
1								(LSB)
2	Reserved							
3	Reserved					PAD LENGTH (000b)		
4	(MSB)			CONTINUATION DESCRIPTOR LENGTH (n-7)				
7								(LSB)
	CDB continuation descriptor type specific data							
8	(MSB)			SOURCE PARTITION_ID				
15								(LSB)
16	(MSB)			SOURCE USER_OBJECT_ID				
23								(LSB)
24	Reserved							CPY_ATTR
25	FREEZE	Reserved			TIME OF DUPLICATION			
26	Reserved							
27								
28	(MSB)			RANGE DESCRIPTORS LENGTH (n-31)				
31								(LSB)
	Range descriptors, if any							
32	Range descriptor [first] (see table x14)							
47								
	⋮							
n-15	Range descriptor [last] (see table x14)							
n								

The CDB CONTINUATION DESCRIPTOR TYPE field contains 0101h (i.e., copy user object source CDB continuation descriptor).

The PAD LENGTH field is set to zero to indicate that no pad bytes are needed to eight byte align a copy user object source CDB continuation descriptor. If the PAD LENGTH field is not set to zero in a copy user object source CDB continuation descriptor, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The CONTINUATION DESCRIPTOR LENGTH field contains the number of bytes that follow in this descriptor.

The SOURCE PARTITION_ID field specifies the Partition_ID (see 4.6.4) of the partition that contains the source user object. If the partition identified by the SOURCE PARTITION_ID field does not exist, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The SOURCE USER_OBJECT_ID field specifies the User_Object_ID of the source user object (see 4.6.5). If the user object identified by the SOURCE USER_OBJECT_ID field does not exist, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The CPY_ATTR (copy attributes) bit specifies whether the attributes from this source user object are copied to the destination user object. If the CPY_ATTR bit is set to zero, no attributes are copied from this source user object to the destination user object. If the CPY_ATTR bit is set to one, all application client settable attributes (see 7.2.1) are copied from this source user object to the destination user object. If the CPY_ATTR bit is set to one, in more than one copy user object source CDB continuation descriptor, attributes copied from one source user object may overwrite attributes copied from another source user object.

If the CPY_ATTR bit is set to one and the reserved data space attribute in the User Object Information attributes page (see 7.1.2.11) of the source user object is set to a value other than zero, the reserved data space attribute in the destination user object shall be increased by the amount of data space reserved for the source user object (i.e., the copy operation shall be treated as appending the source user object to data already in the destination user object).

If the FREEZE bit is set to zero, the copy operation should not modify the contents of the object accessibility attribute in the User Object Information attributes page (see 7.1.2.11) of the source user object. If the FREEZE bit is set to one and source object freeze duplication management is supported (see 4.d.4.2), then the device server shall:

1) Set the object accessibility attribute in the User Object Information attributes page of the source user object to 0000 0001h before starting any copy operations that access the source user object; and
2) Restore the object accessibility attribute in the User Object Information attributes page of the source user object to its previous value after all copy operations that access the source user object are completed.

If the FREEZE bit is set to one and source object freeze duplication management is not supported, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

The TIME OF DUPLICATION field specifies which time of duplication source object management method (see 4.d.4.1) applies to the source user object. If the TIME OF DUPLICATION field is set to DEFAULT (see table x9 in 4.d.4.1), then the default copy user objects time of duplication method attribute in the Partition Information attributes page (see 7.1.2.9) specifies which time of duplication source object management method applies to the source user object.

The RANGE DESCRIPTORS LENGTH field specifies the number of bytes in the range descriptors that follow. If the RANGE DESCRIPTORS LENGTH field is set to zero, the entire source user object shall be appended to (i.e., copied to the end of) the destination user object.

Each range descriptor (see table x14) specifies a number of bytes to copy, a starting byte offset in the source user object, and a starting byte offset in the destination user object.

Table x14 — Range descriptor format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
7				BYTES TO COPY				(LSB)
8	(MSB)							
15				SOURCE BYTE OFFSET				(LSB)
16	(MSB)							
23				DESTINATION BYTE OFFSET				(LSB)

The BYTES TO COPY field indicates the number of bytes to copy for this range descriptor.

The SOURCE BYTE OFFSET field indicates the starting byte offset in the user object specified by the SOURCE PARTITION_ID field and the SOURCE USER_OBJECT_ID field from which bytes are to be read for the copy operation.

The DESTINATION BYTE OFFSET field indicates the starting byte offset in the destination user object to which bytes are to be written by the copy operation. If the DESTINATION BYTE OFFSET field is set to FFFF FFFF FFFF FFFFh, the bytes shall be appended to the destination user object.

The byte ranges specified by individual range descriptors are allowed to overlap (e.g., the second range descriptor may transfer some or all of the same bytes that were transferred by the first range descriptor).

If the values in the BYTES TO COPY field and SOURCE BYTE OFFSET field result an attempt to read a byte that is beyond the user object logical length attribute value in the User Object Information attributes page (see 7.1.2.11) for the user object specified by the SOURCE PARTITION_ID field and the SOURCE USER_OBJECT_ID field, then:

a) The bytes between the user object byte offset and the user object logical length shall be transferred;
b) The command shall be terminated with CHECK CONDITION status, with the sense key shall be set to RECOVERED ERROR and the additional sense code set to READ PAST END OF USER OBJECT;
c) The command-specific information sense data descriptor (see SPC-3) shall be included in the sense data; and
d) The COMMAND-SPECIFIC INFORMATION field shall contain the number of bytes transferred by the command, including but not limited to the bytes transferred by this range descriptor.

…

6.h COPY USER OBJECTS

{{All of 6.h is new. The use of change markups is suspended for the remainder of 6.h.}}

The COPY USER OBJECTS command (see table x15) causes the OSD device server to allocate and initialize one user object and then copy data from one or more source user objects to the newly created user object.

Table x15 — COPY USER OBJECTS command

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)							
9	SERVICE ACTION (8893h)							(LSB)
10	Reserved			DPO	FUA	ISOLATION		
11	Reserved		GET/SET CDBFMT		Reserved			
12	TIMESTAMPS CONTROL							
13	Reserved							
14	DUPLICATION METHOD							
15	Reserved							
16	(MSB)							
23	DESTINATION PARTITION_ID							(LSB)
24	(MSB)							
31	REQUESTED DESTINATION USER_OBJECT_ID							(LSB)
32								
47	Reserved							
48	(MSB)							
51	CDB CONTINUATION LENGTH (see 5.2.x)							(LSB)
52								
79	Get and set attributes parameters (see 5.2.4)							
80								
183	Capability (see 4.11.2.2)							
184								
235	Security parameters (see 5.2.8)							

The contents of the DPO bit and the FUA bit are defined in 5.2.3.

The contents of the ISOLATION field are defined in 5.2.5.

The GET/SET CDBFMT field specifies the format of the get and set attributes parameters as described in 5.2.4.

The contents of the TIMESTAMPS CONTROL field are defined in 5.2.10.

The DUPLICATION METHOD field specifies which duplication method (see 4.d.3) applies to the COPY USER OBJECTS command. If the DUPLICATION METHOD field is set to DEFAULT (see table x8 in 4.d.3), then the default

copy user objects duplication method attribute in the Partition Information attributes page (see 7.1.2.9) specifies which duplication method applies to the COPY USER OBJECTS command.

The contents of the DESTINATION PARTITION_ID field are defined in 5.2.7. If the DESTINATION PARTITION_ID field contains zero, the command shall be terminated with a CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The contents of the REQUESTED DESTINATION USER_OBJECT_ID field specify the User_Object_ID (see 4.6.5) to be assigned to the created user object that is to serve as the destination for the data copies from the source user object or user objects. If the REQUESTED DESTINATION USER_OBJECT_ID field contains zero, any User_Object_ID may be assigned. If the REQUESTED DESTINATION USER_OBJECT_ID field contains any value other than zero and the device server is unable to assign the requested User_Object_ID to the created user object, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

Within a partition, the device server shall not allow:

a) The same User_Object_ID to be associated with more than one user object at any point in time; or
b) A User_Object_ID to have the same value as any assigned Collection_Object_ID.

The contents of the CDB CONTINUATION LENGTH field are defined in 5.2.x. If the CDB CONTINUATION LENGTH field contains zero, the command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

The command shall be terminated with CHECK CONDITION status, with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST, if CDB continuation segment (see 5.x):

a) Does not contain at least one copy user object source CDB continuation descriptor (see 5.y.h);
b) Contains more than one extension capabilities CDB continuation descriptor (see 5.y.z); or
c) Contains any CDB continuation descriptors other than the following:
   A) Copy user object source CDB continuation descriptor (see 5.y.h); and
   B) Extension capabilities CDB continuation descriptor (see 5.y.z).

The copy user object source CDB continuation descriptor or descriptors specify the data that is to be written to the created user object. The copy user object source CDB continuation descriptors shall be processed in the order in which they appear in the CDB continuation segment (i.e., the copy user object source CDB continuation descriptor closest to the beginning of the CDB continuation segment shall be processed first, the next closest descriptor shall be processed second, etc.).

Each copy user object source CDB continuation descriptor may reference the same user object, or a different user object. One capability is necessary for each object accessed, but multiple copy user object CDB continuation descriptors may rely on the capability provided by a single capability.

The get and set attributes parameters are defined in 5.2.4. The format of the Data-In Buffer and Data-Out Buffer when attributes are being retrieved or set is described in 4.14. The User_Object_ID assigned by the COPY USER OBJECTS command may be obtained from the Current Command attributes page (see 7.1.2.29).

The capability is defined in 4.11.2.2. The COPY USER OBJECTS command accesses multiple objects. One capability is necessary for each object accessed. The capability with the highest value (see table 27 in 4.12.4.1) in the SECURITY METHOD field appears in the CDB. The other capabilities appear in the CDB continuation segment (see 5.x).

The security parameters are defined in 5.2.8.

The assigned User_Object_ID shall be placed in the Collection_Object_ID or User_Object_ID attribute in the Current Command attributes page (see 7.1.2.29).

If a COPY USER OBJECTS command causes the value in the number of collections and user objects attribute in the Partition Information attributes page (see 7.1.2.9) to exceed the value in the object count attribute in the Partition Quotas attributes page (see 7.1.2.13), then a quota error shall be generated (see 4.10.2). The quota testing principles described in 4.10.3 apply to the testing of the object count quota.

If a COPY USER OBJECTS command causes the value in the user object logical length attribute in the User Object Information attributes page (see 7.1.2.11) to exceed the value in the maximum user object length attribute in the User Object Quotas attributes page (see 7.1.2.14), then a quota error shall be generated (see 4.10.2). The quota testing principles described in 4.10.3 apply to the testing of the maximum user object length quota.

If a COPY USER OBJECTS command causes the value in the used capacity attribute in the Partition Information attributes page (see 7.1.2.9) to exceed the value in the capacity quota attribute in the Partition Quotas attributes page (see 7.1.2.13), then a quota error shall be generated (see 4.10.2). The quota testing principles described in 4.10.3 apply to the testing of the capacity quota.

…

Table B.1 — Numerical order OSD service action codes

Service Action	Command
…	…
8890h to 8891h	Reserved
8892h	CREATE AND WRITE
8893h	COPY USER OBJECTS
~~8893h to~~ 8894h	Reserved
…	…