To: INCITS Technical Committee T10
From: Fred Knight, Network Appliance, David L. Black, EMC
Email: knight@netapp.com, black_david@emc.com
Date: 14 January, 2009
Subject: SBC-3 Thin Provisioning Commands

1) Revision history

Revision 0 (July 7, 2008) First revision [08-149r0]

Revision 1 (Aug 22, 2008) Revision 1

Split the data path from the management path (create a new proposal for management path). Remove all pools and pool management constructs. This proposal now covers the data path only. Also use the already existing ATA TRIM terminology.

Revision 2 (Sept 5, 2008) Revision 2

Move text from the command clause to the model clause and clarify some language.

Revision 3 (Oct 2, 2008) Revision 3

Introduce LBA state machine with 2 states (hole and allocated) and descriptions of state changes. Remove references to application client and retaining or discarding data, and replace with LBA state changes. Change TRIM back to PUNCH

Revision 4 (Nov 4, 2008) Revision 4

Convert to mapped/unmapped terminology, convert state machine to standard format, redefine read after unmap (previously trim) to match ATA, add PR conflict table entry, CbCS permission table entry, clarify impact of Write Protect on UNMAP command (it fails if write protected), add TPRZ bit (indicating unmapped LBAs read as zero), clarify VERIFY command operation on unmapped LBA, add field to specify max number of extents per UNMAP command to the blocks VPD page, and account for XCOPY (SPC).

Revision 5 (Nov 17, 2008)

Create unmap operation used by UNMAP command, create lists from several long sentences, propose specific ASC/Q values, update the PRE-FETCH command, add the REASSIGN BLOCKS command behavior for unmapped LBAs, clarify the VERIFY command.

Revision 6 (Nov 20, 2008)

Incorporate comments from 11/19 con-call. Make the threshold crossing Unit Attention a 1 time event (until specific external events occur to reset it, and allow another 1 time event); to solve the problem of bouncing around the threshold point; add VERIFY table for BYTCHK operation.

Revision 7 (Dec 5, 2008)

Incorporate comments from 12/4 con-call. Extract threshold events into a separate proposal. Since verify operations of unmapped LBAs do not retrieve data (BYTCHK=0 doesn't retrieve data, and BYTCHK=1 is illegal), change all references to data retrieved during read or verify to just data retrieved during read.

Revision 8 (Jan 8, 2009)

Complete review changes of "LBAs in the [un]mapped state" to "[un]mapped LBAs", and removal of retrieving data during verify. Change TPE bit = 0 from "fully provisioned" to "not thin as defined in this standard".

Revision 9 (Jan 14, 2009)

Incorporate changes from CAP meeting.

Revision 10 [08-356r5] (Jan 15, 2009)

Specify effects of an unmap operation on an unmapped LBA. Merge in and rewrite UNMAP bit for support for WRITE SAME (16) and (32) support from 08-356 (David Black), including model clause material on performing an unmap operation instead of a write operation. Resubmit as 08-356r5 with the more general document name.

2) *Related documents*

spc4r16 – SCSI Primary Commands – 4
sbc3r15 – SCSI Block Commands – 3
09-011r0 – Thin Provisioning Threshold Notification
08-356r4 – WRITE SAME unmap bit
08-341r0 – Thin Provisioning Management Commands
T13/e07154r6 – ATA8-ACS2 TRIM proposal (accepted)
T13/e08137r4 – ATA8-ACS2 DRAT – Deterministic Read After Trim (accepted).

3) *Overview*

Traditional storage devices pre-allocate physical storage for every possible logical block. There is a fixed one-to-one relationship between the physical storage and the logical storage (every logical block is permanently mapped to a physical block). Generally speaking, the physical capacity of the device is always the same as the logical capacity of the device (plus spares if any). The READ

CAPACITY command reports the usable number of logical blocks to the application client. Historically, this has been referred to simply as the capacity of the device. These devices are fully provisioned.

Thinly provisioned devices also report the capacity in the READ CAPACITY command, but they do not allocate (or map) their physical storage in the same way that fully provisioned devices do. Thinly provisioned devices do not necessarily have a permanent one-to-one relationship between the physical storage and the logical storage. Thinly provisioned devices may report a different capacity (in the READ CAPACITY command), than their actual physical capacity. These devices often report a larger capacity than the actual physical capacity for storing user data.

One typical use of storage is a creation and deletion process. Files are created, possibly modified, and saved as new files (with the old one being deleted). Databases are created, where records are added, updated, and deleted.

Fully provisioned storage must allocate space to retain all possible data represented by every block described by the logical capacity (their physical capacity must be the same (or greater) than their reported logical capacity). These devices are always capable of receiving write data into the pre-defined and pre-allocated space.

Thinly provisioned devices may or may not pre-allocate space to retain write data. When a write is received, physical storage may be allocated from a pool of available storage to retain the write data and a mapping established between the location of that physical storage and the appropriate location within the logical capacity (a physical to LBA mapping). As long as this allocation and mapping process is successful, the write operates in the same way that it does on a fully provisioned storage device. However, if all the available physical capacity has been used, and no space can be allocated to retain the write data, the write operation must fail. This failure must have a new unique ASCQ.

In addition, to aid application clients it is desired to notify the client before it actually reaches the point when the failure occurs. These may return a UNIT ATTENTION if the I/O needs to be retried to succeed. These are new types of status conditions to return to the application client, and as such need new ASCQ values to define these conditions. This is needed as part of the I/O path so that error recovery can be synchronized. Out of band techniques would not enable the needed synchronization for error recovery. For example, the application may be involved in notification of a storage administrator to take corrective action, or explicitly taking corrective action of its own. To synchronize that action with the I/O requires this be part of the I/O path.

Event	Sense Key	NEW ASC/Qs	ASC/Q
Temporary lack of physical	NOT READY	LOGICAL UNIT NOT	04/xx

blocks – write not done, retry required		READY, SPACE ALLOCATION IN PROCESS	
Persistent lack of physical blocks – write not done, retry will not help	DATA PROTECT	SPACE ALLOCATION FAILED WRITE PROTECT	27/xx

Note: ASC/Q values above are suggestions to start discussion; the final values are to be assigned by the editor.

When the host no longer needs to retain the data (such as when a host file is deleted, or a database record is deleted), there is no specific action required by a fully provisioned device. However, a thinly provisioned device may benefit by knowing about this event and be able to return the physical blocks containing this “deleted” data to a pool of available blocks. Since the data has been deleted, the storage device need not retain the contents of those blocks. If those LBAs are accessed by an application client (a READ is done), the storage device would be free to return any particular data (zeros, -1, etc). This “delete” function is done via the UNMAP command.

Other possible use cases exist for individual disks, SSD devices, backplane RAID controllers and external RAID controllers.

This proposal defines commands and error codes for the operation of thinly provisioned devices. The UNMAP command includes a method to supply a list of extent descriptors (LBA and length) to a device server. An UNMAP bit is also added to the WRITE SAME (16) and WRITE SAME (32) commands; see 08-356r4 for overview and rationale that applies to this bit.

Management functions will be presented in a separate proposal.

Existing text is shown in **BLACK**, new text is shown in **RED**, and comments (not to be included) are shown in **BLUE**.

Proposal:

3.1.22 format corrupt: a vendor-specific condition in which the application client may not be able to perform read operations, write operations, unmap operations, or verify operations. See 4.7.

<...>

3.1.32 logical unit reset: A condition resulting from the events defined by SAM-4 in which the logical unit performs the logical unit reset operations described in SAM-4, this standard, and other applicable command standards (see table 13 in 5.1).

3.1.32a mapped: A state of an LBA in which there exists a known relationship to a physical block.

3.1.33 media: Plural of medium.

<…>

3.1.54 unit attention condition: A state that a logical unit (see 3.1.30) maintains while the logical unit has asynchronous status information to report to the initiator ports associated with one or more I_T nexuses (see 3.1.25). See SAM-4.

3.1.54a unmapped: A state of an LBA in which the relationship to a physical block is not defined.

3.1.55 unrecovered error: An error for which a device server is unable to read or write a logical block within the recovery limits specified in the Read-Write Error Recovery mode page (see 6.3.5) and the Verify Error Recovery mode page (see 6.3.6).

4 Direct-access block device type model

4.1 Direct-access block device type model overview

SCSI devices that conform to this standard are referred to as direct-access block devices. This includes the category of logical units commonly referred to as rigid disks and removable rigid disks. MMC-4 is typically used by CD-ROM devices.

This standard is intended to be used in conjunction with SAM-4, SPC-4, SCC-2, SES-2, and SMC-2.

Direct-access block devices store data for later retrieval in logical blocks. Logical blocks contain user data, may contain protection information accessible to the application client, and may contain additional information not normally accessible to the application client (e.g., an ECC). The number of bytes of user data contained in each logical block is the logical block length. The logical block length is greater than or equal to one byte and should be even. Most direct-access block devices support a logical block length of 512 bytes and some support additional logical block lengths (e.g., 520 or 4096 bytes). The logical block length does not include the length of protection information and additional information, if any, that are associated with the logical block. The logical block length is the same for all logical blocks on the medium.

Each logical block is stored at a unique LBA, which is either four bytes (i.e., a short LBA) or eight bytes (i.e., a long LBA) in length. The LBAs on a logical unit shall begin with zero and shall be contiguous up to the last logical block on the logical unit. The LBAs may be mapped to physical blocks or unmapped (see 4.4.1). An application client uses commands performing write operations to store logical blocks and commands performing read operations to retrieve logical blocks. A write operation causes one or more logical blocks to be written to the medium. A read operation of mapped logical blocks causes one or more logical blocks to be read from the medium. A read operation may also cause data for unmapped logical blocks to be retrieved (see 4.4.1.3). A verify operation confirms that one or more logical blocks were correctly written and are able to be read without error ~~from the medium~~. An unmap operation causes a change in the relationship between LBAs and physical blocks and may cause a change in the data returned by a subsequent read operation.

Logical blocks are stored by a process that causes localized changes or transitions within a medium. The changes made to the medium to store the logical blocks may be volatile (i.e., not retained through power cycles) or non-volatile (i.e., retained through power cycles). The medium may contain vendor-specific information that is not addressable through an LBA. Such data may include defect management data and other device management information.

<…>

4.3 Removable medium

4.3.1 Removable medium overview

The medium may be removable or non-removable. The removable medium may be contained within a cartridge or jacket to prevent damage to the recording surfaces.

A removable medium has an attribute of being mounted or unmounted on a suitable transport mechanism in a direct-access block device. A removable medium is mounted when the direct-access block device is capable of performing write, read, unmap, and verify operations to the medium. A removable medium is unmounted at any other time (e.g., during loading, unloading, or storage).

An application client may check whether a removable medium is mounted by issuing a TEST UNIT READY command (see SPC-4). A direct-access block device containing a removable medium may not be accessible for write, read, unmap, and verify operations until it receives a START STOP UNIT command (see 5.19).

If the direct-access block device implements cache, either volatile or non-volatile, it ensures that all logical blocks of the medium contain the most recent user data and protection information, if any, prior to permitting unmounting of the removable medium.

If the medium in a direct-access block device is removable, and the medium is removed, then the device server shall establish a unit attention condition with the additional sense code set to the appropriate value (e.g., NOT READY TO READY CHANGE, MEDIUM MAY HAVE CHANGED).

<…>

4.4 Logical Blocks

Logical blocks are stored on the medium along with additional information that the device server uses to manage storage and retrieval. The format of the additional information is defined by other standards or is vendor-specific and is hidden from the application client during normal read, write, unmap, and verify operations. This additional information may be used to identify the physical location of the blocks of data, the address of the logical block, the relationship between mapped LBAs and physical blocks, and to provide protection against the loss of user data and protection information, if any (e.g., by containing ECC bytes).

The first LBA is zero. The last LBA is [n-1], where [n] is the number of logical blocks on the medium accessible by the application client. The READ CAPACITY (10) parameter data (see 5.12.2 and 5.13.2) RETURNED LOGICAL BLOCK ADDRESS field indicates the value of [n-1].

LBAs are no larger than 8 bytes. Some commands support only 4-byte (i.e., short) LOGICAL BLOCK ADDRESS fields (e.g., READ CAPACITY (10), READ (10), and WRITE (10)). If the capacity exceeds that accessible with short LBAs, then the device server returns a capacity of FFFF_FFFFh in response to a READ CAPACITY (10) command, indicating that:

a) the application client should enable descriptor format sense data (see SPC-4) in the Control mode page (see SPC-4) and in any REQUEST SENSE commands (see SPC-4) it sends; and

b) the application client should use commands with 8-byte LOGICAL BLOCK ADDRESS fields (e.g., READ CAPACITY (16), READ (16), and WRITE (16)).

NOTE 2 - If a command with a 4-byte LOGICAL BLOCK ADDRESS field accesses logical blocks beyond LBAs FFFF_FFFFh and fixed format sense data is used, there is no field in the sense data large enough to report the LBA of an error (see 4.14).

If a command is received that references or attempts to access a logical block not within the capacity of the medium, then the device server terminates the command with CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to LOGICAL BLOCK ADDRESS OUT OF RANGE. The device server may terminate the command before processing or after the device server has transferred some or all of the data.

The number of bytes of user data contained in a logical block is the logical block length. The parameter data returned by the device server in response to a READ CAPACITY command (see 5.12) describes the logical block length that is used on the medium. The mode parameter block descriptor (see 6.3.2) is used by an application client to change the logical block length in direct-access block devices that support changeable logical block lengths. The logical block length should be used to determine does not include the length of protection information and additional information, if any.

The location of a logical block on the medium is not required to have a relationship to the location of any other logical block. However, in a typical direct-access block device, the time to access a logical block at LBA [x+1] after accessing LBA [x] is often less than the time to access some other logical block. The time to access the logical block at LBA [x] and then the logical block at LBA [x+1] need not be less than time to access LBA [x] and then LBA [x+100]. The READ CAPACITY command issued with a PMI bit set to one may be useful in determining where longer access times occur.

4.4.1 Logical Block Provisioning

4.4.1.1 Provisioning Overview

Each LBA may be mapped or unmapped. For LBAs that are mapped, there is a known relationship to a physical block. For LBAs that are unmapped, the relationship between an LBA and a physical block is not defined. Figure A shows the relationships of LBAs to physical blocks in these two states.

LBA 0	LBA 1	LBA 2	LBA 3	LBA 4	LBA 5	LBA 6	LBA 7
PB	PB	UNMAPPED	PB	UNMAPPED	UNMAPPED	PB	PB

LBA 0	LBA 1	LBA 2	LBA 3	LBA 4	LBA 5	LBA 6	LBA 7
PB		UNMAPPED		PB		UNMAPPED	

Key:
LBA n = logical block with LBA n
PB = a unique physical block
UNMAPPED = the relationship to a physical block is not defined

Figure A – Mapped and Unmapped Logical Blocks

Transitions between the mapped and unmapped state are shown in figure B.

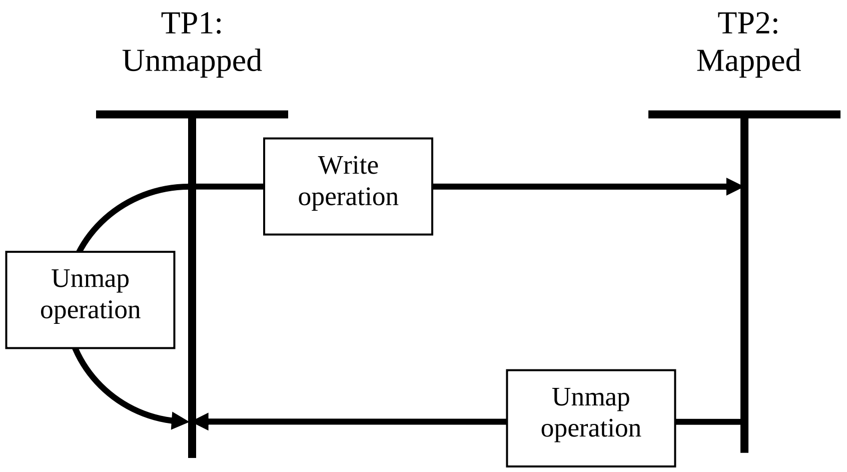


Figure B – Logical Block State Transitions

4.4.1.2 TP2: Mapped state

When in the mapped state, a relationship exists between the LBA and a physical block. User data and protection information read without error from a logical block in the mapped state shall be the user data and protection information that was most recently written to that LBA. If data has not been written to that LBA, user data and protection information read from the LBA shall be the user data and protection information that was established during initialization (see 4.7).

4.4.1.2.1 Transition TP2:Mapped state to TP1:Unmapped state

This transition shall occur when an unmap operation completes without error on a mapped LBA (see 4.4.1.5.2). As a result of this transition, the user data retrieved by the next read operation may be indeterminate unless otherwise specified (e.g., see the TPRZ bit in READ CAPACITY (16) parameter data (6.4.3)).

If this transition occurs as the result of an operation that transfers data for the logical block from the data-out buffer (e.g., a WRITE SAME command with the UNMAP bit set to one), then the user data and protection information, if any, retrieved during a read operation for the LBA after the transition shall be the same as if a write operation to that LBA had completed without error using that data.

4.4.1.3 TP1: Unmapped state

When in the unmapped state, no relationship is defined between the LBA and a physical block (see figure A). For an unmapped LBA, user data and protection information retrieved during a read operation:

a) shall not be retrieved from data that was previously written to any other LBA; and
b) shall not change after the user data and protection information, if any, has been sent to the application client until a subsequent write operation to that LBA occurs (i.e., all read operations to an LBA shall retrieve the same data until a subsequent write operation to that LBA occurs).

If protection information is enabled, the device server shall use a default value of FFFF_FFFF_FFFF_FFFFh as the protection information for unmapped LBAs.

If the TPRZ bit is set to one in the READ CAPACITY (16) parameter data (see 5.13.2), then for an unmapped LBA, the user data retrieved during a read operation shall have all bits set to zero.

4.4.1.3.1 Transition TP1:Unmapped state to TP2:Mapped state

This transition:

a) shall occur when a write operation occurs. As part of processing a write operation, the device server may cause LBAs other than those specified by that write operation to transition from unmapped state to mapped state; or
b) may occur at any other time for vendor specific reasons.

For each LBA that transitions from the unmapped state to the mapped state without transferring data for the logical block from the data-out buffer, the contents of the logical block after the transition shall be the same as if a read operation of the unmapped LBA had been followed by a write operation to that LBA using the data retrieved by that read operation.

4.4.1.3.2 Transition TP1:Unmapped state to TP1:Unmapped state

This transition shall occur when an unmap operation completes without error on an unmapped LBA (see 4.4.1.5.2). As a result of this transition, the user data retrieved by the next read operation may be indeterminate unless otherwise specified (e.g., see the TPRZ bit in READ CAPACITY (16) parameter data (6.4.3)).

If this transition occurs as the result of an operation that transfers data for the logical block from the data-out buffer (e.g., a WRITE SAME command with the UNMAP bit set to one), then the user data and protection information, if any, retrieved during a read operation for the LBA after the transition shall be the same as if a write operation to that LBA had completed without error using that data.

4.4.1.4 Full provisioning

A fully provisioned logical unit ensures that a sufficient number of physical blocks are available to retain user data for all logical blocks within the logical unit’s reported capacity (see 5.12 - READ CAPACITY (10) and 5.13 - READ CAPACITY(16)). Fully provisioned logical units provide a known relationship between all logical blocks and physical blocks (see figures 2 and 3).

The initial state of all LBAs on a fully provisioned logical unit is mapped. LBAs on fully provisioned logical units shall not transition out of the mapped state.

Support for full provisioning is indicated by the TPE bit in the READ CAPACITY (16) parameter data.

4.4.1.5 Thin Provisioning

4.4.1.5.1 Thin Provisioning Overview

A thin provisioned logical unit may report a capacity (see 5.12 - READ CAPACITY (10) and 5.13 - READ CAPACITY (16)) larger than the number of mapped LBAs. A thin provisioned logical unit may or may not have sufficient physical blocks to retain user data transferred as a result of a write operation and storing the write data on the medium.

The initial state of each LBA on a thin provisioned logical unit is unmapped (see figure B).

A device server that supports thin provisioning may also support one or more soft threshold values. The method of setting the soft threshold values is outside the scope of this standard.

Support for thin provisioning is indicated by the TPE bit in the READ CAPACITY (16) parameter data. Logical units implementing thin provisioning:

a) Shall set the TPE bit to one in the READ CAPACITY(16) parameter data (see 6.4.3); and
b) Shall support at least one of the following unmap mechanisms:
    a. The UNMAP command (see 5.x.1);
    b. The UNMAP bit in the WRITE SAME (16) CDB (see 5.38); or
    c. The UNMAP bit in the WRITE SAME (32) CDB (see 5.39).

A device server that supports the UNMAP command shall:
a) Set the MAXIMUM UNMAP LBA COUNT field in the BLOCK LIMITS VPD page (see 6.4.3) to a value greater than or equal to one; and
b) Set the MAXIMUM UNMAP DESCRIPTOR COUNT field in the BLOCK LIMITS VPD page (see 6.4.3) to a value greater than or equal to one.

.

4.4.1.5.2 Unmap Operation

An unmap operation transitions one or more LBAs to the unmapped state. More than one physical block may be affected by an unmapped operation. The data in all other mapped logical blocks on the medium shall be preserved. Performing an unmap operation on an unmapped LBA shall not be considered as an error (see 4.4.1.3.2).

A command may request that a device server should perform an unmap operation instead of a write operation (e.g., a WRITE SAME, with the UNMAP bit set to one). If a device server is unable to perform such a requested unmap operation on an LBA in accordance with the requirements of this standard, then the device server shall instead perform the write operation specified by the command on that LBA, and this shall not be considered as an error. The device server shall perform the write operation and not perform the unmap operation, if the TPRZ bit is set to one in the READ CAPACITY (16) parameter data (see 6.4.3) and:
a) Any bit in the user data transferred from the data out buffer is not zero; or
b) The protection information, if any, transferred from the data out buffer is not set to FFFF_FFFF_FFFF_FFFFh.

4.4.1.5.3 Thin Provisioning and Protection Information

If protection information is enabled, the protection information for LBAs in the:

a) mapped state shall be as described in 4.17; and
b) unmapped state shall be as described in 4.4.1.3.

4.4.1.5.4 Resource Exhaustion Considerations

If a write operation is received and a temporary lack of physical block resources prevents the logical unit from storing the write data on the medium, then the device server shall terminate the command with the sense key set to NOT READY and the additional sense code set to SPACE ALLOCATION IN PROCESS. The recommended application client recovery action is to issue the command again at a later time.

If a write operation is received and a persistent lack of physical block resources prevents the logical unit from storing the write data on the medium, then the device server shall terminate the command with the sense key set to DATA PROTECT and the additional sense code set to SPACE ALLOCATION FAILED WRITE PROTECT. This condition shall not cause the WP bit in the DEVICE-SPECIFIC PARAMETER field (see 6.3.1) of the mode page header to be set to one. Application client recovery actions for this status are outside the scope of this standard.

4.5 Physical blocks

A physical block is a set of data bytes on the medium accessed by the device server as a unit. A physical block may contain:

a) a portion of a logical block (i.e., there are multiple physical blocks in the logical block)(e.g., a physical block length of 512 bytes with a logical block length of 2 048 bytes);
b) a single complete logical block; or
c) more than one logical block (i.e., there are multiple logical blocks in the physical block)(e.g., a physical block length of 4 096 bytes with a logical block length of 512 bytes).

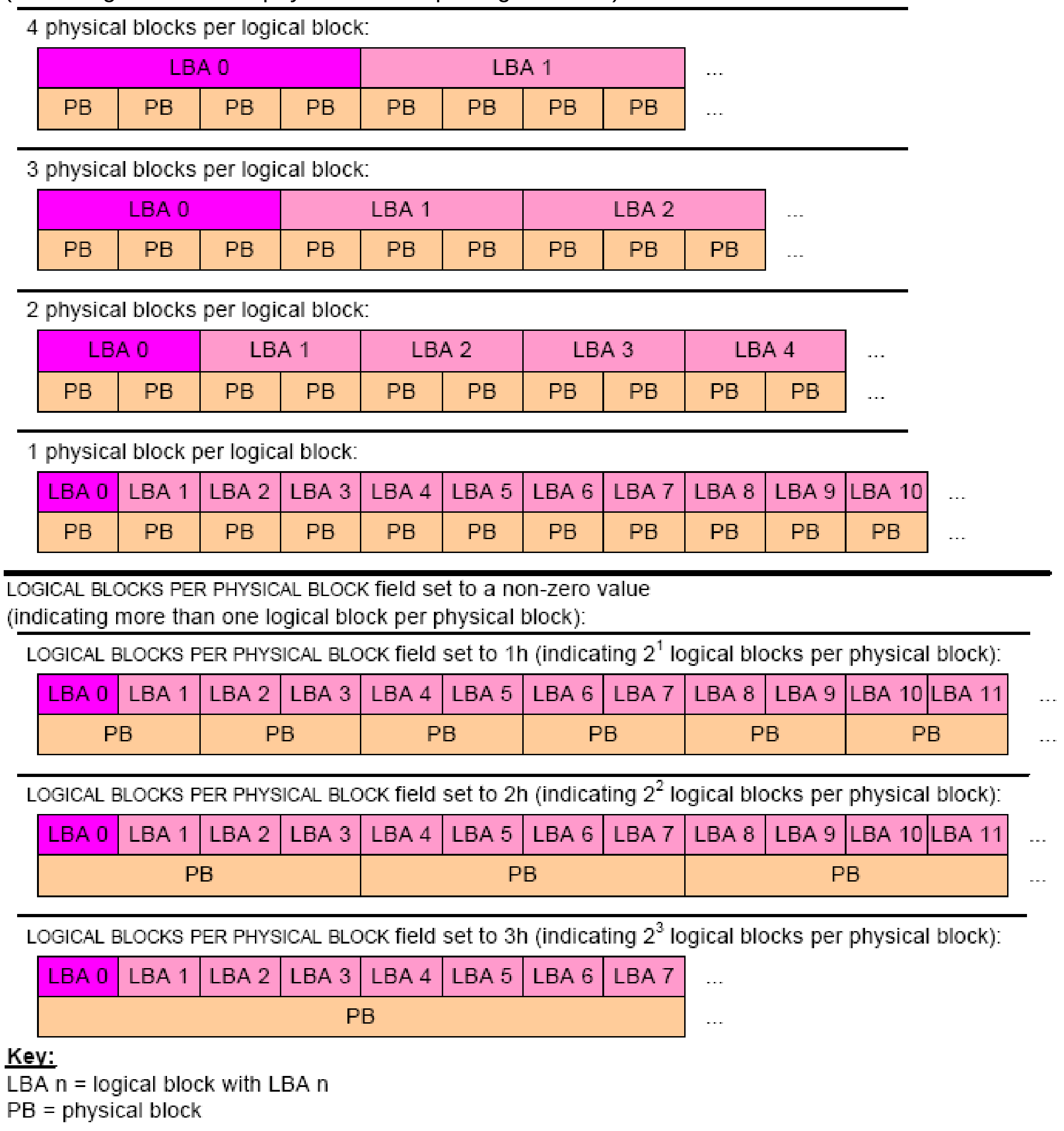
Each physical block includes additional information not normally accessible to the application client (e.g., an ECC) that the device server uses to manage storage and retrieval.

If the device server supports the COR_DIS bit and/or the WR_UNCOR bit in a WRITE LONG command (see 5.35 and 5.36), then the device server shall have the capability of marking individual logical blocks as containing pseudo uncorrectable errors with correction enabled (see 3.1.45) or with correction disabled (see 3.1.46).

Logical blocks may or may not be aligned to physical block boundaries. A mechanism for establishing the alignment is not defined by this standard.

Figure 2 shows examples of logical blocks and physical blocks, where LBA 0 is aligned to a physical block boundary.

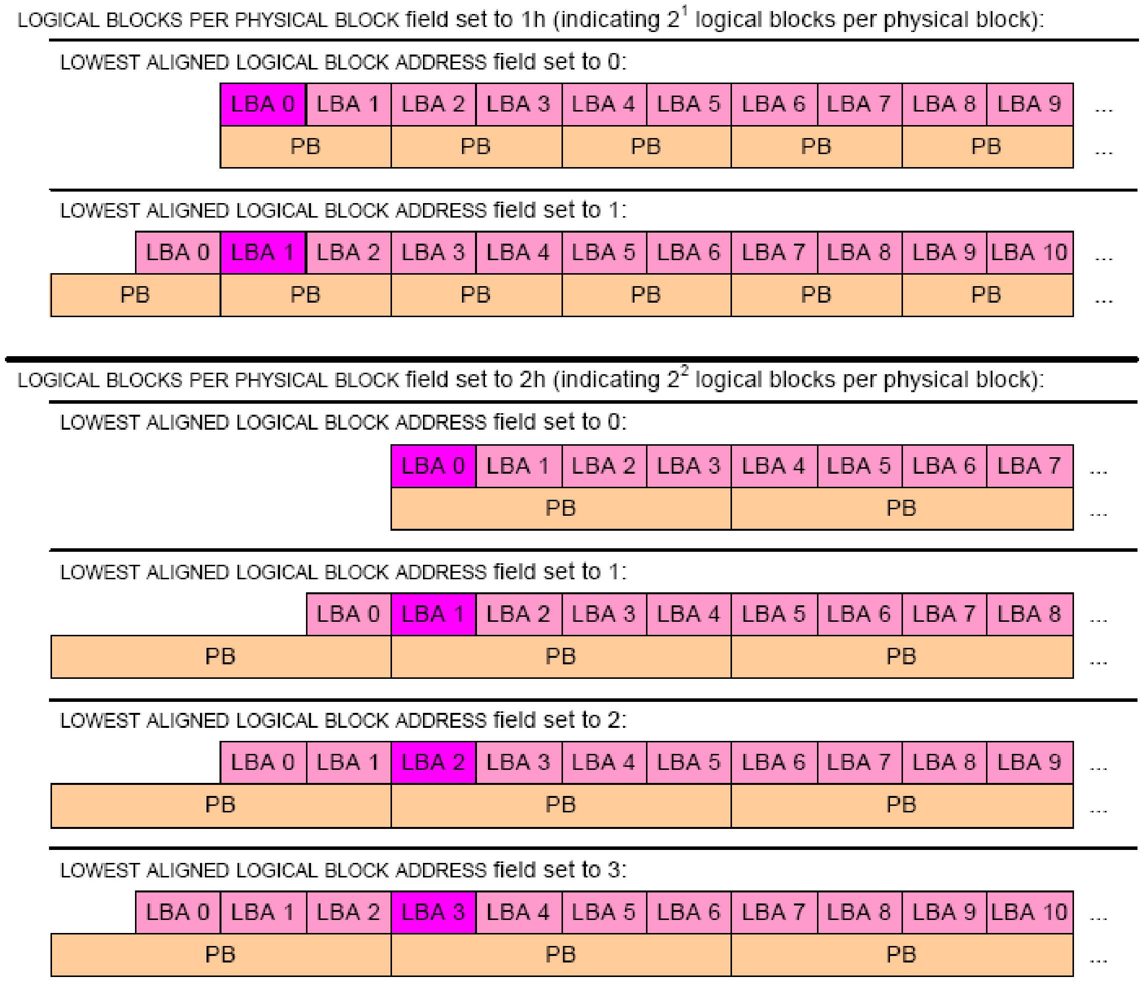
LOGICAL BLOCKS PER PHYSICAL BLOCK field (see 5.13.2) set to 0h (indicating one or more physical blocks per logical block):



4 physical blocks per logical block:
LBA 0
LBA 1
PB
...
3 physical blocks per logical block:
LBA 0
LBA 1
LBA 2
PB
...
2 physical blocks per logical block:
LBA 0
LBA 1
LBA 2
LBA 3
LBA 4
PB
...
1 physical block per logical block:
LBA 0
LBA 1
LBA 2
LBA 3
LBA 4
LBA 5
LBA 6
LBA 7
LBA 8
LBA 9
LBA 10
PB
...
LOGICAL BLOCKS PER PHYSICAL BLOCK field set to a non-zero value (indicating more than one logical block per physical block):
LOGICAL BLOCKS PER PHYSICAL BLOCK field set to 1h (indicating 2^1 logical blocks per physical block):
LBA 11
PB
LOGICAL BLOCKS PER PHYSICAL BLOCK field set to 2h (indicating 2^2 logical blocks per physical block):
PB
LOGICAL BLOCKS PER PHYSICAL BLOCK field set to 3h (indicating 2^3 logical blocks per physical block):
PB
Key:
LBA n = logical block with LBA n
PB = physical block

Figure 2 — Logical blocks and physical blocks examples

Figure 3 shows examples of logical blocks and physical blocks, where various LBAs are aligned to the physical block boundaries.


LOGICAL BLOCKS PER PHYSICAL BLOCK field set to 1h (indicating 2^1 logical blocks per physical block):
LOWEST ALIGNED LOGICAL BLOCK ADDRESS field set to 0:
LBA 0 LBA 1 LBA 2 LBA 3 LBA 4 LBA 5 LBA 6 LBA 7 LBA 8 LBA 9 ...
PB PB PB PB PB ...
LOWEST ALIGNED LOGICAL BLOCK ADDRESS field set to 1:
LBA 0 LBA 1 LBA 2 LBA 3 LBA 4 LBA 5 LBA 6 LBA 7 LBA 8 LBA 9 LBA 10 ...
PB PB PB PB PB PB ...
LOGICAL BLOCKS PER PHYSICAL BLOCK field set to 2h (indicating 2^2 logical blocks per physical block):
LOWEST ALIGNED LOGICAL BLOCK ADDRESS field set to 0:
LBA 0 LBA 1 LBA 2 LBA 3 LBA 4 LBA 5 LBA 6 LBA 7 ...
PB PB ...
LOWEST ALIGNED LOGICAL BLOCK ADDRESS field set to 1:
LBA 0 LBA 1 LBA 2 LBA 3 LBA 4 LBA 5 LBA 6 LBA 7 LBA 8 ...
PB PB PB ...
LOWEST ALIGNED LOGICAL BLOCK ADDRESS field set to 2:
LBA 0 LBA 1 LBA 2 LBA 3 LBA 4 LBA 5 LBA 6 LBA 7 LBA 8 LBA 9 ...
PB PB PB ...
LOWEST ALIGNED LOGICAL BLOCK ADDRESS field set to 3:
LBA 0 LBA 1 LBA 2 LBA 3 LBA 4 LBA 5 LBA 6 LBA 7 LBA 8 LBA 9 LBA 10 ...
PB PB PB ...

Key:
LBA n = logical block with LBA n
PB = physical block

Figure 3 — Logical block to physical block alignment examples

When there are more than one logical block per physical block, not all of the logical blocks are aligned to the physical block boundaries. When using medium access commands, application clients should:

a) specify an LBA that is aligned to a physical block boundary; and
b) access an integral number of physical blocks, provided that the access does not go beyond the last LBA on the medium.

4.6 Ready state

A direct-access block device is ready when the device server is capable of processing medium access commands (i.e., commands that perform read operations, write operations, unmap operations, or verify operations).

A direct-access block device using removable media is not ready until a volume is mounted and other conditions are met (see 4.2). If a direct-access block device is not ready, then the device server shall terminate medium access commands with CHECK CONDITION status with the sense key set to NOT READY and the appropriate additional sense code for the condition.

Some direct-access block devices may be switched from being ready to being not ready by using the START STOP UNIT command (see 5.19). An application client may need to issue a START STOP UNIT command with a START bit set to one to make a direct-access block device ready.

4.7 Initialization

Direct-access block devices may require initialization prior to write, read, unmap, and verify operations. This initialization is performed by a FORMAT UNIT command (see 5.2). Parameters related to the format (e.g., logical block length) may be set with the MODE SELECT command prior to the format operation. Some direct-access block devices are initialized by means not specified in this standard. The time when the initialization occurs is vendor-specific.

Direct-access block devices using a non-volatile medium may save the parameters and only need to be initialized once. However, some mode parameters may need to be initialized after each logical unit reset. A catastrophic failure of the direct-access block device may require the FORMAT UNIT command to be issued.

Direct-access block devices that use a volatile medium may need to be initialized after each logical unit reset prior to the processing of write, read, unmap, or verify operations. Mode parameters may also need initialization after logical unit resets.

NOTE 3 - Mode parameter block descriptors read with the MODE SENSE command before a FORMAT UNIT completes may contain information that may not reflect the true state of the medium.

A direct-access block device may become format corrupt after processing a MODE SELECT command that changes parameters related to the medium format. During this time, the device server may terminate medium access commands with CHECK CONDITION status with the sense key set to NOT READY and the appropriate additional sense code for the condition.

Any time the parameter data to be returned by a device server in response to a READ CAPACITY (10) command (see 5.12) or a READ CAPACITY (16) command (see 5.13) changes (e.g., when a FORMAT UNIT command or a MODE SELECT command completes changing the number of logical blocks, logical block length, protection information, or reference tag ownership values, or when a vendor-specific mechanism causes a change), then the device server shall establish a unit attention condition for the SCSI initiator port (see SAM-4) associated with each I_T nexus, except the I_T nexus on which the command causing the change was received, with the additional sense code set to CAPACITY DATA HAS CHANGED.

NOTE 4 - Logical units compliant with previous versions of this standard were not required to establish a unit attention condition.

4.8 Write protection

Write protection prevents the alteration of the medium by commands issued to the device server. Write protection is usually controlled by the user of the medium through manual intervention (e.g., mechanical lock) or may result from hardware controls (e.g., tabs on the media housing) or software write protection. All sources of write protection are independent. When present, any write protection shall cause otherwise valid commands that request alteration of the medium to be rejected with CHECK CONDITION status with the sense key set to DATA PROTECT. Only when all write protections are disabled shall the device server process unmap operations, or commands that request alteration of the medium.

Hardware write protection results when a physical attribute of the drive or medium is changed to specify that writing shall be prohibited. Changing the state of the hardware write protection requires physical intervention, either with the drive or the medium. If allowed by the drive, changing the hardware write protection while the medium is mounted results in vendor-specific behavior that may include the writing of previously buffered data (e.g., data in cache).

Software write protection results when the device server is marked as write protected by the application client using the SWP bit in the Control mode page (see SPC-4). Software write protection is optional. Changing the state of software write protection shall not prevent previously accepted data (e.g., data in cache) from being written to the media.

The device server reports the status of write protection in the device server and on the medium with the DEVICE-SPECIFIC PARAMETER field in the mode parameter header (see 6.3.1).

<...>

4.10 Write failures

If one or more commands performing write operations are in the task set and are being processed when power is lost (e.g., resulting in a vendor-specific command timeout by the application client) or a medium error or hardware error occurs (e.g., because a removable medium was incorrectly unmounted), then the:

a) mapped state of the LBA (see 4.4.1) may be indeterminate, if thin provisioning is supported; and
b) data in the logical blocks being written by those commands is indeterminate.

When accessed by a command performing a read or verify operation (e.g., after power on or after the removable medium is mounted), the device server may return old data, new data, or vendor-specific data in those logical blocks.

Before reading or verifying logical blocks which encountered such a failure, an application client should reissue any commands performing write operations that were outstanding.

4.11 Caches

Direct-access block devices may implement caches. A cache is an area of temporary storage in the direct-access block device with a fast access time that is used to enhance performance. Cache exists separately from the medium and is not directly accessible by the application client. Use of cache for write or read operations may reduce the access time to a logical block and increase the overall data throughput.

<...>

During read operations, the device server uses the cache to store logical blocks that the application client may request at some future time. The algorithm used to manage the cache is not part of this standard. However, parameters are provided to advise the device server about future requests, or to restrict the use of cache for a particular request.

During write operations, the device server uses the cache to store data that is to be written to the medium at a later time. This is called write-back caching. The command may complete prior to logical blocks being written to the medium. As a result of using a write-back caching there is a period of time when the data may be lost if power to the SCSI target device is lost and a volatile cache is being used or a hardware failure occurs. There is also the possibility of an error occurring during the subsequent write operation. If an error occurred during the write operation, it may be reported as a deferred error on a later command. The application client may request that write-back caching be disabled with the Caching mode page (see 6.3.4) to prevent detected write errors from being reported as deferred errors. Even with write-back caching disabled, undetected write errors may occur. The VERIFY commands and the WRITE AND VERIFY commands may be used to detect those errors.

During unmap operations, the device server updates cache to prevent retrieval of stale data during a subsequent read operation.

When the cache becomes full of logical blocks, new logical blocks may replace those currently in the cache. The disable page out (DPO) bit in the CDB of commands performing write, read, or verify operations allows the application client to influence the replacement of logical blocks in the cache. For write operations, setting the DPO bit to one specifies that the device server should not replace existing logical blocks in the cache with the new logical blocks being written. For read and verify operations, setting the DPO bit to one specifies that the device server should not replace logical blocks in the cache with the logical blocks that are being read.

NOTE 5 - This does not mean that stale data is allowed in the cache. If a write operation accesses the same LBA as a logical block in the cache, the logical block in the cache is updated with the new write data. If an unmap operation accesses the same LBA as a logical block in the cache, the logical block in the cache is updated with the new read data (see 4.1.3) or removed from the cache.

<…>

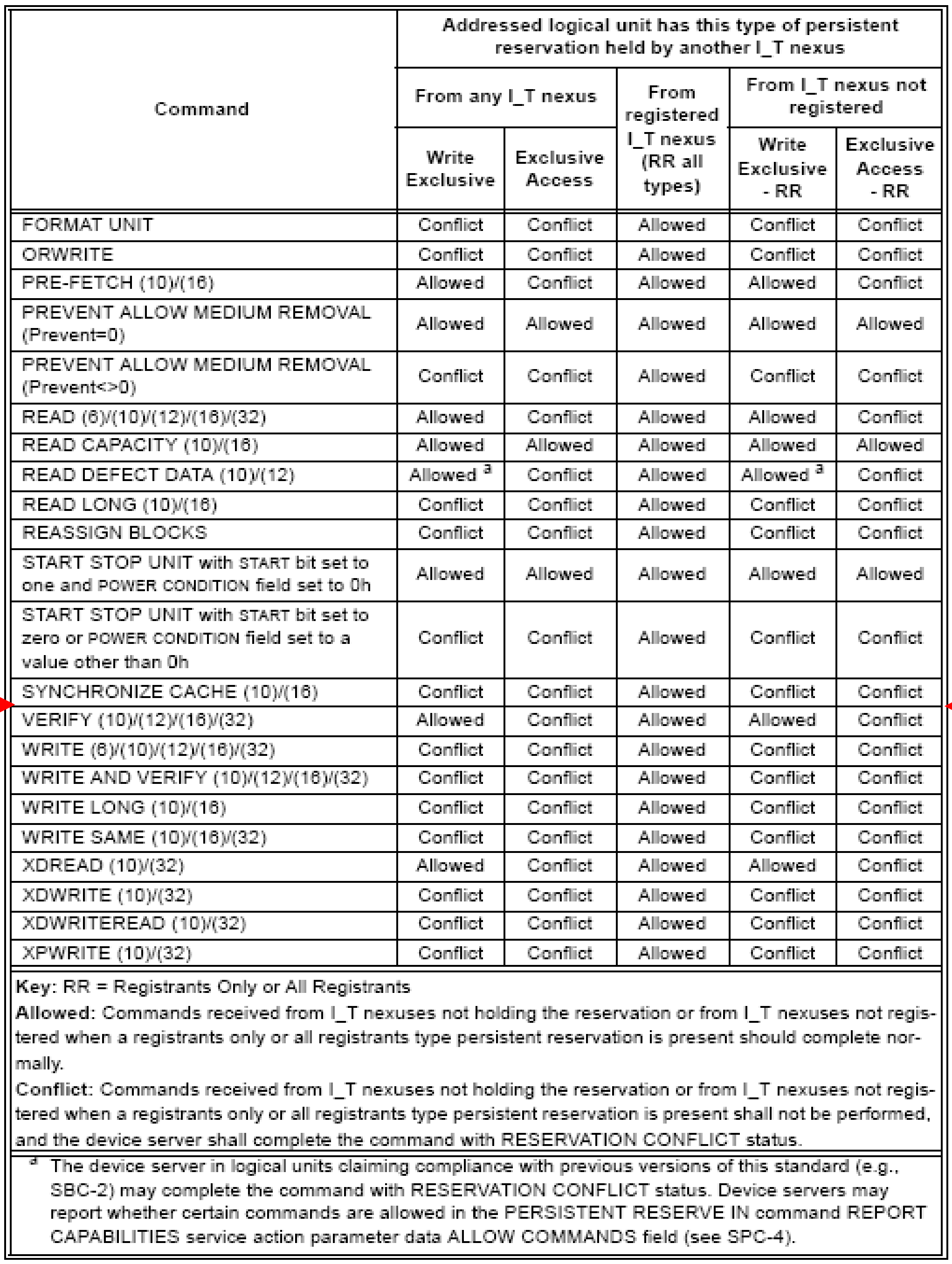
4.13 RESERVATIONS

Reservation restrictions are placed on commands as a result of access qualifiers associated with the type of reservation. See SPC-4 for a description of reservations. The details of commands that are allowed under what types of reservations are described in table 3.

Commands from I_T nexuses holding a reservation should complete normally. Table 3 specifies the behavior of commands from registered I_T nexuses when a registrants only or all registrants type persistent reservation is present.

For each command, this standard or SPC-4 defines the conditions that result in the device server completing the command with RESERVATION CONFLICT status.

Table 3 — SBC-2 commands that are allowed in the presence of various reservations



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 I_T nexus not registered	
	Write Exclusive	Exclusive Access		Write Exclusive - RR	Exclusive Access - RR
FORMAT UNIT	Conflict	Conflict	Allowed	Conflict	Conflict
ORWRITE	Conflict	Conflict	Allowed	Conflict	Conflict
PRE-FETCH (10)/(16)	Allowed	Conflict	Allowed	Allowed	Conflict
PREVENT ALLOW MEDIUM REMOVAL (Prevent=0)	Allowed	Allowed	Allowed	Allowed	Allowed
PREVENT ALLOW MEDIUM REMOVAL (Prevent<>0)	Conflict	Conflict	Allowed	Conflict	Conflict
READ (6)/(10)/(12)/(16)/(32)	Allowed	Conflict	Allowed	Allowed	Conflict
READ CAPACITY (10)/(16)	Allowed	Allowed	Allowed	Allowed	Allowed
READ DEFECT DATA (10)/(12)	Allowed [a]	Conflict	Allowed	Allowed [a]	Conflict
READ LONG (10)/(16)	Conflict	Conflict	Allowed	Conflict	Conflict
REASSIGN BLOCKS	Conflict	Conflict	Allowed	Conflict	Conflict
START STOP UNIT with START bit set to one and POWER CONDITION field set to 0h	Allowed	Allowed	Allowed	Allowed	Allowed
START STOP UNIT with START bit set to zero or POWER CONDITION field set to a value other than 0h	Conflict	Conflict	Allowed	Conflict	Conflict
SYNCHRONIZE CACHE (10)/(16)	Conflict	Conflict	Allowed	Conflict	Conflict
VERIFY (10)/(12)/(16)/(32)	Allowed	Conflict	Allowed	Allowed	Conflict
WRITE (6)/(10)/(12)/(16)/(32)	Conflict	Conflict	Allowed	Conflict	Conflict
WRITE AND VERIFY (10)/(12)/(16)/(32)	Conflict	Conflict	Allowed	Conflict	Conflict
WRITE LONG (10)/(16)	Conflict	Conflict	Allowed	Conflict	Conflict
WRITE SAME (10)/(16)/(32)	Conflict	Conflict	Allowed	Conflict	Conflict
XDREAD (10)/(32)	Allowed	Conflict	Allowed	Allowed	Conflict
XDWRITE (10)/(32)	Conflict	Conflict	Allowed	Conflict	Conflict
XDWRITEREAD (10)/(32)	Conflict	Conflict	Allowed	Conflict	Conflict
XPWRITE (10)/(32)	Conflict	Conflict	Allowed	Conflict	Conflict

Key: RR = Registrants Only or All Registrants
Allowed: Commands received from I_T nexuses not holding the reservation or from I_T nexuses not registered when a registrants only or all registrants type persistent reservation is present should complete normally.
Conflict: Commands received from I_T nexuses not holding the reservation or from I_T nexuses not registered when a registrants only or all registrants type persistent reservation is present shall not be performed, and the device server shall complete the command with RESERVATION CONFLICT status.

[a] The device server in logical units claiming compliance with previous versions of this standard (e.g., SBC-2) may complete the command with RESERVATION CONFLICT status. Device servers may report whether certain commands are allowed in the PERSISTENT RESERVE IN command REPORT CAPABILITIES service action parameter data ALLOW COMMANDS field (see SPC-4).

Add UNMAP command matching WRITE (6/10/12/16/32) -

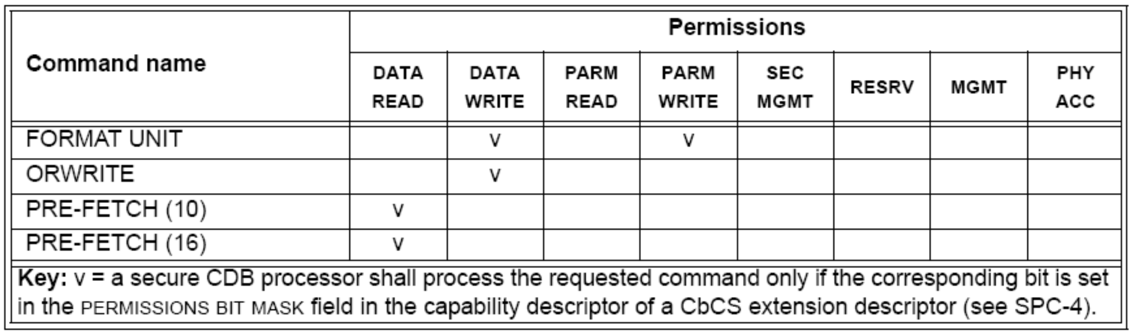
UNMAP	Conflict	Conflict	Allowed	Conflict	Conflict

<...>

4.20 Association between commands and CbCS permission bits

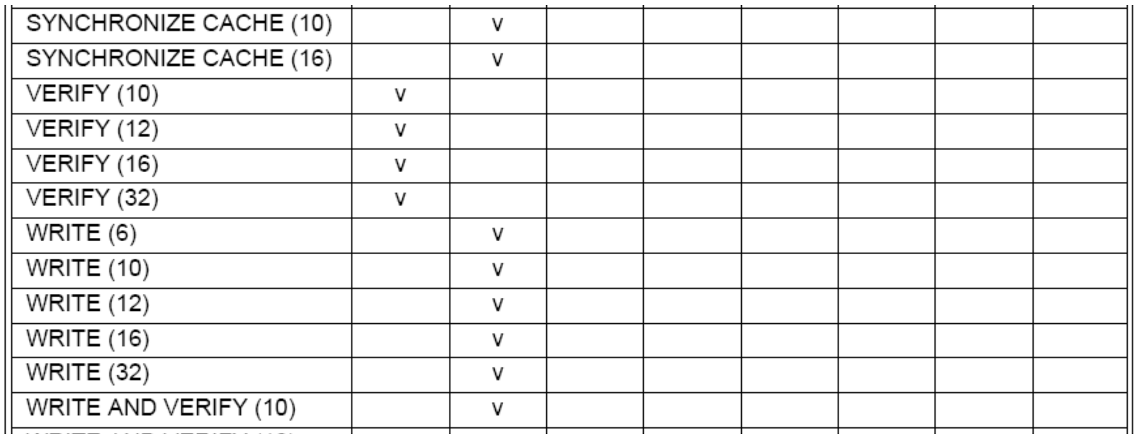
Table 12 defines the Capability based Command Security (i.e., CbCS) permissions required for each command defined in this standard to be processed by a secure CDB processor. The permissions shown in table 12 are defined in the PERMISSIONS BIT MASK field in the capability descriptor of a CbCS extension descriptor (see SPC-4). This standard does not define any permission specific to block commands.

Table 12 — Associations between commands and CbCS permissions (part 1 of 3)



Command name	Permissions							
	DATA READ	DATA WRITE	PARM READ	PARM WRITE	SEC MGMT	RESRV	MGMT	PHY ACC
FORMAT UNIT		v		v				
ORWRITE		v						
PRE-FETCH (10)	v							
PRE-FETCH (16)	v							

Key: v = a secure CDB processor shall process the requested command only if the corresponding bit is set in the PERMISSIONS BIT MASK field in the capability descriptor of a CbCS extension descriptor (see SPC-4).



SYNCHRONIZE CACHE (10)		v						
SYNCHRONIZE CACHE (16)		v						
VERIFY (10)	v							
VERIFY (12)	v							
VERIFY (16)	v							
VERIFY (32)	v							
WRITE (6)		v						
WRITE (10)		v						
WRITE (12)		v						
WRITE (16)		v						
WRITE (32)		v						
WRITE AND VERIFY (10)		v						

Add UNMAP command as follows (DATA WRITE permissions required):

UNMAP	v

<...>

5 Commands for direct-access block devices

5.1 Commands for direct-access block devices overview

The commands for direct-access block devices are listed in table 13. Commands with CDB or parameter data fields that support protection information (see 4.17) or for which protection information may be a factor in the processing of the command are indicated by the fourth (i.e, Protection information) column.

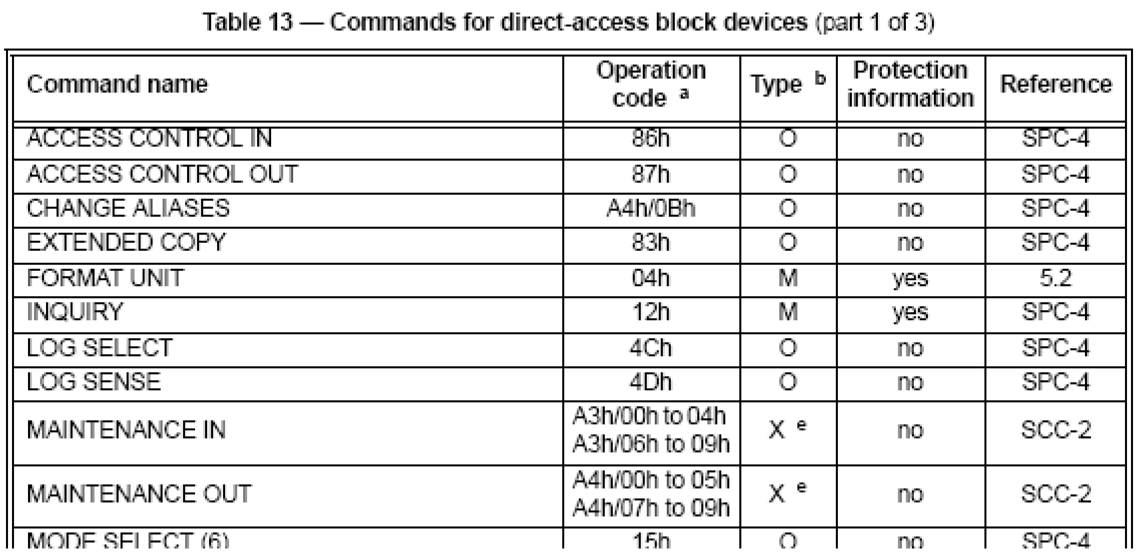
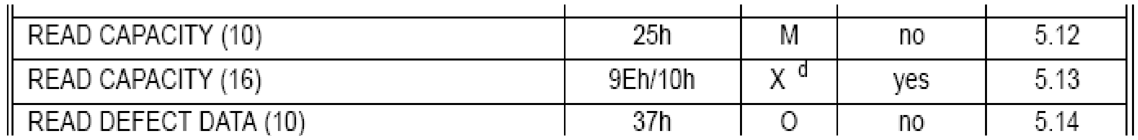


Table 13 — Commands for direct-access block devices (part 1 of 3)

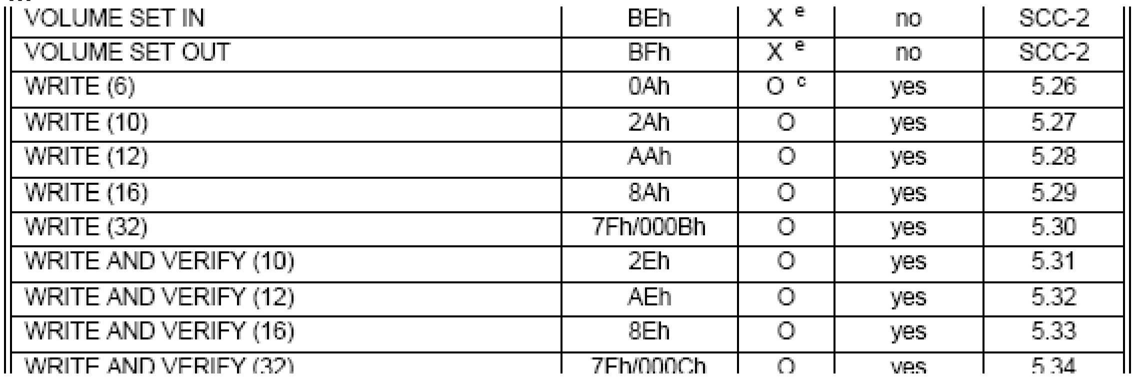
Command name	Operation code [a]	Type [b]	Protection information	Reference
ACCESS CONTROL IN	86h	O	no	SPC-4
ACCESS CONTROL OUT	87h	O	no	SPC-4
CHANGE ALIASES	A4h/0Bh	O	no	SPC-4
EXTENDED COPY	83h	O	no	SPC-4
FORMAT UNIT	04h	M	yes	5.2
INQUIRY	12h	M	yes	SPC-4
LOG SELECT	4Ch	O	no	SPC-4
LOG SENSE	4Dh	O	no	SPC-4
MAINTENANCE IN	A3h/00h to 04h A3h/06h to 09h	X [e]	no	SCC-2
MAINTENANCE OUT	A4h/00h to 05h A4h/07h to 09h	X [e]	no	SCC-2
MODE SELECT (6)	15h	O	no	SPC-4

<...>



READ CAPACITY (10)	25h	M	no	5.12
READ CAPACITY (16)	9Eh/10h	X [d]	yes	5.13
READ DEFECT DATA (10)	37h	O	no	5.14

<...>



VOLUME SET IN	BEh	X [e]	no	SCC-2
VOLUME SET OUT	BFh	X [e]	no	SCC-2
WRITE (6)	0Ah	O [c]	yes	5.26
WRITE (10)	2Ah	O	yes	5.27
WRITE (12)	AAh	O	yes	5.28
WRITE (16)	8Ah	O	yes	5.29
WRITE (32)	7Fh/000Bh	O	yes	5.30
WRITE AND VERIFY (10)	2Eh	O	yes	5.31
WRITE AND VERIFY (12)	AEh	O	yes	5.32
WRITE AND VERIFY (16)	8Eh	O	yes	5.33
WRITE AND VERIFY (32)	7Fh/000Ch	O	yes	5.34

Add UNMAP command as follows:

UNMAP	42h	O(g)	yes	5.x

(d) READ CAPACITY (16) is mandatory if protection information or thin provisioning (see 4.4.1.5) is supported and optional otherwise.

<...>

(g) See 4.4.1.5.

Add footnote (g) to the O entry for Type for the WRITE SAME (16) and WRITE SAME (32) commands.

<...>

5.2 FORMAT UNIT command

5.2.1 FORMAT UNIT command overview

The FORMAT UNIT command (see table 15) requests that the device server format the medium into application client accessible logical blocks as specified in the number of logical blocks and logical block length values received in the last mode parameter block descriptor (see 6.3.2) in a MODE SELECT command (see SPC-4). In addition, the device server may certify the medium and create control structures for the management of the medium and defects. The degree that the medium is altered by this command is vendor-specific.

If a device server receives a FORMAT UNIT command before receiving a MODE SELECT command with a mode parameter block descriptor, then the device server shall use the number of logical blocks and logical block length at which the logical unit is currently formatted (i.e., no change is made to the number of logical blocks and the logical block length of the logical unit during the format operation).

If any deferred downloaded code has been received as a result of a WRITE BUFFER command (see SPC-4), then that deferred downloaded code shall replace the current operational code.

Table 15 — FORMAT UNIT command

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (04h)							
1	FMTPINFO		LONGLIST	FMTDATA	CMPLST	DEFECT LIST FORMAT		
2	Vendor-specific							
3	Obsolete							
4								
5	Control							

The simplest form of the FORMAT UNIT command (i.e., a FORMAT UNIT command with no parameter data) accomplishes medium formatting with little application client control over defect management. The device server implementation determines the degree of defect management that is to be performed. Additional forms of this command increase the application client's control over defect management. The application client may specify:

a) defect list(s) to be used;
b) defect locations;
c) that logical unit certification be enabled; and
d) exception handling in the event that defect lists are not accessible.

While performing a format operation, the device server shall terminate all commands except INQUIRY commands, REPORT LUNS commands, and REQUEST SENSE commands with CHECK CONDITION status with the sense key set to NOT READY and the additional sense code set to LOGICAL UNIT NOT READY, FORMAT IN PROGRESS. If the device server receives an INQUIRY command, a REPORT LUNS commands, or a REQUEST SENSE command, then the device server shall process the command. The device server shall return data for an INQUIRY command based on the condition of the SCSI target device before beginning the FORMAT UNIT command (i.e., INQUIRY data shall not change until after successful completion of a format operation). The processing of commands in the task set when a FORMAT UNIT command is received is vendor-specific.

The PROGRESS INDICATION field in parameter data returned by the device server in response to a REQUEST SENSE command (see SPC-4) may be used by the application client at any time during a format operation to poll the logical unit's progress. While a format operation is in progress unless an error has occurred, a device server shall respond to a REQUEST SENSE command by returning parameter data containing sense data with the sense key set to NOT READY and the additional sense code set to LOGICAL UNIT NOT READY, FORMAT IN PROGRESS with the sense key specific bytes set for progress indication (see SPC-4).

The OPERATION CODE field is defined in SPC-4 and shall be set to the value defined in table 15.

The format protection information (FMTPINFO) field (see table 20) in combination with the PROTECTION FIELD USAGE field (see 5.2.2.2) specifies whether or not the device server enables or disables the use of protection information.

Following a successful format, the P_TYPE field in the READ CAPACITY (16) parameter data (see 5.13.1) indicates the type of protection currently in effect on the logical unit. LBAs on a:

a) fully provisioned logical unit (see 4.4.1.4) shall be mapped; or
b) thin provisioned logical unit (see 4.4.1.5) shall be unmapped.

When protection information is written during a FORMAT UNIT command (i.e., the FMTPINFO field is set to a value greater than zero), protection information shall be written to a default value of FFFF_FFFF_FFFF_FFFFh. A LONGLIST bit set to zero specifies that the parameter list, if any, contains a short parameter list header as defined in table 18. A LONGLIST bit set to one specifies that the parameter list, if any, contains a long parameter list header as defined in table 19. If the FMTDATA bit is set to zero, then the LONGLIST bit shall be ignored.

A format data (FMTDATA) bit set to zero specifies that no parameter list be transferred from the data-out buffer.

A FMTDATA bit set to one specifies that the FORMAT UNIT parameter list (see table 17) shall be transferred from the data-out buffer. The parameter list consists of a parameter list header, followed by an optional initialization pattern descriptor, followed by an optional defect list.

<…>

5.4 PRE-FETCH (10) command

The PRE-FETCH (10) command (see table 32) requests that the device server:

a) for any mapped LBAs, transfer the specified logical blocks from the medium to the volatile cache and/or non-volatile cache; and
b) for any unmapped LBAs, update the volatile cache and/or non-volatile cache to prevent retrieval of stale data.

Logical blocks include user data and, if the medium is formatted with protection information enabled, protection information. No data shall be transferred to the data-in buffer.

The OPERATION CODE field is defined in SPC-4 and shall be set to the value defined in table 32.

<…>

5.5 PRE-FETCH (16) command

The PRE-FETCH (16) command (see table 33) requests that the device server:

a) for any mapped LBAs, transfer the specified logical blocks from the medium to the volatile cache and/or non-volatile cache; and
b) for any unmapped LBAs, update the volatile cache and/or non-volatile cache to prevent retrieval of stale data.

Logical blocks include user data and, if the medium is formatted with protection information enabled, protection information. No data shall be transferred to the data-in buffer.

<...>

5.7 READ (6) command

The READ (6) command (see table 36) requests that the device server read the specified logical block(s) and transfer them to the data-in buffer. Each logical block read includes user data and, if the medium is formatted with protection information enabled, protection information. Each logical block transferred includes user data but does not include protection information. ~~The most recent data value written, or to be written, if cached, in the addressed logical blocks shall be returned.~~

The OPERATION CODE field is defined in SPC-4 and shall be set to the value defined in table 36.

<...>

5.8 READ (10) command

The READ (10) command (see table 38) requests that the device server read the specified logical block(s) and transfer them to the data-in buffer. Each logical block read includes user data and, if the medium is formatted with protection information enabled, protection information. Each logical block transferred includes user data and may include protection information, based on the RDPROTECT field and the medium format. ~~The most recent data value written in the addressed logical block shall be returned.~~

<...>

5.13 READ CAPACITY (16) command

5.13.1 READ CAPACITY (16) command overview

The READ CAPACITY (16) command (see table 46) requests that the device server transfer parameter data describing the capacity and medium format of the direct-access block device to the data-in buffer. This command is mandatory if the logical unit supports protection information (see 4.17) and is optional otherwise. This command is implemented as a service action of the SERVICE ACTION IN operation code (see A.2). This command may be processed as if it has a HEAD OF QUEUE task attribute (see 4.12).

Table 46 – READ CAPACITY (16) command

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (9Eh)							
1	Reserved			Service Action (10h)				
2	(MSB)							
	LOGICAL BLOCK ADDRESS							
9								(LSB)
10	(MSB)							
	ALLOCATION LENGTH							
13								(LSB)
14	Reserved							PMI
15	Control							

The OPERATION CODE field and SERVICE ACTION field are defined in SPC-4 and shall be set to the values defined in table 46.

See the READ CAPACITY (10) command (see 5.12) for definitions of the LOGICAL BLOCK ADDRESS field and the PMI bit.

The ALLOCATION LENGTH field specifies the maximum number of bytes that the application client has allocated for returned parameter data. An allocation length of zero indicates that no data shall be transferred. This condition shall not be considered as an error. The device server shall terminate transfers to the data-in buffer when the number of bytes specified by the ALLOCATION LENGTH field have been transferred or when all available data has been transferred, whichever is less. The contents of the parameter data shall not be altered to reflect the truncation, if any, that results from an insufficient allocation length.

The contents of the CONTROL byte are defined in SAM-4.

5.13.2 READ CAPACITY (16) parameter data

The READ CAPACITY (16) parameter data is defined in table 47. Any time the READ CAPACITY (16) parameter data changes, the device server should establish a unit attention condition as described in 4.7.

Table 47 – READ CAPACITY (16) parameter data

Byte \ Bit	7	6	5	4	3	2	1	0
0 – 7	(MSB) RETURNED LOGICAL BLOCK ADDRESS (LSB)							
8 – 11	(MSB) LOGICAL BLOCK LENGTH IN BYTES (LSB)							
12	Reserved				P_TYPE			PROT_EN
13	Reserved				LOGICAL BLOCKS PER PHYSICAL BLOCK EXPONENT			
14	TPE	TPRZ	(MSB) LOWEST ALIGNED LOGICAL BLOCK ADDRESS					
15	LOWEST ALIGNED LOGICAL BLOCK ADDRESS (LSB)							
16 – 31	Reserved							

The RETURNED LOGICAL BLOCK ADDRESS field and LOGICAL BLOCK LENGTH IN BYTES field of the READ CAPACITY (16) parameter data are the same as the in the READ CAPACITY (10) parameter data (see 5.12). The maximum value that shall be returned in the RETURNED LOGICAL BLOCK ADDRESS field is FFFF_FFFF_FFFF_FFFEh.

The protection type (P_TYPE) field and the protection enable (PROT_EN) bit (see table 48) indicate the logical unit's current type of protection.

Table 48 — P_TYPE field and PROT_EN bit

PROT_EN	P_TYPE	Description
0	xxxb	The logical unit is formatted to type 0 protection (see 4.17.2.2).
1	000b	The logical unit is formatted to type 1 protection (see 4.17.2.3).
1	001b	The logical unit is formatted to type 2 protection (see 4.17.2.4).
1	010b	The logical unit is formatted to type 3 protection (see 4.17.2.5).
1	011b - 111b	Reserved

The LOGICAL BLOCKS PER PHYSICAL BLOCK EXPONENT field is defined in table 49.

Table 49 — LOGICAL BLOCKS PER PHYSICAL BLOCK EXPONENT field

Code	Description
0	One or more physical blocks per logical block [a]
n > 0	2^n logical blocks per physical block

[a] The number of physical blocks per logical block is not reported.

A thin provisioning enabled (TPE) bit set to one indicates that the logical unit implements thin provisioning (see 4.4.1.5).

A TPE bit set to zero indicates that the logical unit implements full provisioning (see 4.4.1.4).

A thin provisioning read zeros (TPRZ) bit set to one indicates that a read operation of an unmapped LBA shall retrieve user data with all bits set to zero.

A TPRZ bit set to zero indicates that a read operation of an unmapped LBA may retrieve user data with bits of other than all zeros.

The LOWEST ALIGNED LOGICAL BLOCK ADDRESS field indicates the LBA of the first logical block that is located at the beginning of a physical block (see 4.5).

NOTE 14 - The highest LBA that the lowest aligned logical block address field supports is 3FFFh (i.e., 16383).

<...>

5.18 REASSIGN BLOCKS command

5.18.1 REASSIGN BLOCKS command overview

The REASSIGN BLOCKS command (see table 56) requests that the device server reassign defective logical blocks to another area on the medium set aside for this purpose. The device server should also record the location of the defective logical blocks in the GLIST, if supported. This command shall not alter the contents of the PLIST (see 4.9).

The parameter list provided in the data-out buffer contains a defective LBA list that contains the LBAs of the logical blocks to be reassigned. The device server shall reassign the parts of the medium used for each logical block in the defective LBA list. More than one physical block may be relocated by each LBA. If the device server is able to recover user data and protection information, if any, from the original logical block, then the device server shall write the recovered user data and any protection information to the ~~reassigned logical block~~ LBA that was reassigned. If the LBA is in the unmapped state, the device server shall transition the LBA to the mapped state and write the data retrieved during a read operation (see 4.4.1.3) to the LBA that was reassigned. If the device server is unable to recover user data and protection information, if any, then the device server shall write vendor-specific data as the user data and shall write a default value of FFFF_FFFF_FFFF_FFFFh as the protection information, if enabled. The data in all other logical blocks on the medium shall be preserved.

<...>

5.20 SYNCHRONIZE CACHE (10) command

The SYNCHRONIZE CACHE (10) command (see table 63) requests that the device server ensure that the specified logical blocks have their most recent data values recorded in non-volatile cache and/or on the medium, based on the SYNC_NV bit. Logical blocks include user data and, if the medium is formatted with protection information enabled, protection information. Logical blocks may or may not be removed from volatile cache and non-volatile cache as a result of the synchronize cache operation.

<...>

5.38 WRITE SAME (16) command

The WRITE SAME (16) command (see table 90) requests that the device server transfer a single logical block from the data-out buffer and write the contents of that logical block, with modifications based on the LBDATA bit and the PBDATA bit, to the specified range of LBAs. Each logical block includes user data and may include protection information, based on the WRPROTECT field and the medium format.

Add an UNMAP bit to the CDB:

Table 90 – WRITE SAME (16) Command

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (93h)							
1	WRPROTECT			**Reserved**	~~**Reserved**~~ **UNMAP**	PBDATA	LBDATA	Reserved
2 – 9	(MSB) LOGICAL BLOCK ADDRESS (LSB)							
10 – 13	(MSB) NUMBER OF LOGICAL BLOCKS (LSB)							
14	Reserved			GROUP NUMBER				
15	CONTROL							

The OPERATION CODE field is defined in SPC-4 and shall be set to the value defined in table 90.

If the UNMAP bit is set to one, and the logical unit is thin provisioned (see 4.4.1.5), then the device server should perform an unmap operation on each specified LBA instead of a write operation (see 4.4.1.5.2). If the UNMAP bit is set to zero or the logical unit is not thin provisioned, then the UNMAP bit shall be ignored.

See the WRITE SAME (10) command (see 5.37) for the definitions of the other fields in this command.

5.39 WRITE SAME (32) command

The WRITE SAME (32) command (see table 91) requests that the device server transfer a single logical block from the data-out buffer and write the contents of that logical block, with modifications based on the LBDATA bit and the PBDATA bit, to the specified range of LBAs. Each logical block includes user data and may include protection information, based on the WRPROTECT field and the medium format.

Add an UNMAP bit to the CDB:

Table 91 – WRITE SAME (32) Command

Byte \ Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (7Fh)							
1	CONTROL							
2 to 5	Reserved							
6	Reserved			GROUP NUMBER				
7	ADDITIONAL CDB LENGTH (18h)							
8	(MSB) SERVICE ACTION (000Dh)							
9								(LSB)
10	WRPROTECT			**Reserved**	~~**Reserved**~~ **UNMAP**	PBDATA	LBDATA	Obsolete
11	Reserved							
12 to 31	**No change to remainder of CDB fields**							

The OPERATION CODE field, ADDITIONAL CDB LENGTH field, and SERVICE ACTION field are defined in SPC-4 and shall be set to the values defined in table 91.

See the WRITE SAME (10) command (see 5.37) for the definitions of the CONTROL byte, GROUP NUMBER field, the WRPROTECT field, the PBDATA bit, the LBDATA bit, the LOGICAL BLOCK ADDRESS field, and the NUMBER OF LOGICAL BLOCKS field. See the WRITE SAME (16) command (see 5.38) for the definition of the UNMAP bit.

No further changes in this section.

5. x UNMAP command

5. x. 1 UNMAP command overview

The UNMAP command requests that the device server should transition one or more logical blocks to the unmapped state. The UNMAP command shall be implemented by device servers supporting thin provisioning (see 4.4.1.5).

Table x.1 – UNMAP Command

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (42h)							
1	Reserved							
2	Reserved							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved			GROUP NUMBER				
7	(MSB) PARAMETER LIST LENGTH							
8								(LSB)
9	Control							

The OPERATION CODE field is defined in SPC-4 and shall be set to the value defined in table x.1.

See the PRE-FETCH (10) command (see 5.4) and 4.18 for the definition of the GROUP NUMBER field.

The PARAMETER LIST LENGTH field specifies the length in bytes of the UNMAP PARAMETER LIST that shall be transferred from the application client to the device server. A PARAMETER LIST LENGTH of zero specifies that no data shall be transferred.

The contents of the CONTROL byte are defined in SAM-4.

5. x. 2 UNMAP parameter list

The UNMAP parameter list (see table x.3) contains a parameter list header followed by a UNMAP DESCRIPTOR LIST containing one or more UNMAP DESCRIPTOR fields.

Table x.3 – UNMAP parameter list

Byte \ Bit	7	6	5	4	3	2	1	0
0 – 1	UNMAP DESCRIPTOR LIST LENGH (n-3)							
2 – 7	Reserved							
	UNMAP DESCRIPTOR LIST							
8 – 23	UNMAP DESCRIPTOR [first]							
…	…							
n-15 – n	UNMAP DESCRIPTOR [last]							

The UNMAP DESCRIPTOR LIST LENGTH describes the length of the UNMAP DESCRIPTOR LIST.

The UNMAP DESCRIPTOR LIST contains a list of LBA extents to be operated on. The UNMAP DESCRIPTOR is described in table x.4. The LBAs in the UNMAP DESCRIPTOR LIST may contain overlapping extents, and may be in any order.

For each specified LBA:
a) a mapped LBA should be unmapped, or may remain mapped; and
b) an unmapped LBA shall remain unmapped.

Table x.4 – UNMAP DESCRIPTOR

Byte \ Bit	7	6	5	4	3	2	1	0
0 – 7	(MSB) LOGICAL BLOCK ADDRESS (LSB)							
8 – 11	(MSB) NUMBER OF LOGICAL BLOCKS (LSB)							
12 – 15	RESERVED							

If the LBA plus the number of logical blocks exceeds the capacity of the medium, then the device server shall terminate the command with CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to LOGICAL BLOCK ADDRESS OUT OF RANGE.

If the total number of logical blocks represented in the UNMAP DESCRIPTOR LIST exceeds the value of the MAXIMUM UNMAP LBA COUNT field (see 6.4.2), or if the number of UNMAP DESCRIPTORs exceeds the value of the MAXIMUM UNMAP DESCRIPTOR COUNT field (see 6.4.2), then the device server shall terminate the command with CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN PARAMETER LIST.

A number of logical blocks set to zero shall not be considered an error.

<...>

5.22 VERIFY (10) command

The VERIFY (10) command (see table 66) requests that the device server verify the specified logical block(s) on the medium. Each logical block includes user data and may include protection information, based on the VRPROTECT field and the medium format.

Table 66 — VERIFY (10) command

Byte \ Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (2Fh)							
1	VRPROTECT			DPO	Reserved		BYTCHK	Obsolete
2	(MSB)	LOGICAL BLOCK ADDRESS						
5								(LSB)
6	Restricted for MMC-6	Reserved		GROUP NUMBER				
7	(MSB)	VERIFICATION LENGTH						
8								(LSB)
9	CONTROL							

Logical units that contain cache shall write referenced cached logical blocks to the medium for the logical unit (e.g., as they would do in response to a SYNCHRONIZE CACHE command (see 5.20 and 5.21) with the SYNC_NV bit set to zero, the LOGICAL BLOCK ADDRESS field set to the value of the VERIFY command's LOGICAL BLOCK ADDRESS field, and the NUMBER OF BLOCKS field set to the value of the VERIFY command's VERIFICATION LENGTH field).

The OPERATION CODE field is defined in SPC-4 and shall be set to the value defined in table 66.

See the READ (10) command (see 5.8) for the definition of the DPO bit. See the PRE-FETCH (10) command (see 5.4) for the definition of the LOGICAL BLOCK ADDRESS field. See the PRE-FETCH (10) command (see 5.4) and 4.18 for the definition of the GROUP NUMBER field.

If the Verify Error Recovery mode page (see 6.3.6) is implemented, then the current settings in that page specify the verification criteria. If the Verify Error Recovery mode page is not implemented, then the verification criteria is vendor-specific.

For each referenced LBA, operation of the BYTCHK field is defined in table 66.Y.

Table 66.Y – BYTCHK Operation

	BYTCHK = 0		BYTCHK = 1	
LBA State	Mapped	Unmapped	Mapped	Unmapped
Perform media verification	Yes	No[a]	Yes	Illegal [c]
Transfer user data from the data-out buffer	No	No	Yes	
Comparing of transferred user data	None	None	Perform a byte-by byte comparison of user data read from the medium and user data transferred from the data-out buffer	
Handling of Protection Information, if enabled.				
Transfer user data and protection information from data-out buffer	No	No	Yes	Illegal[c]
Checking of protection information read from the medium	Based on the VRPROTECT field (see table 67)	None[b]	Based on the VRPROTECT field (see table 68)	
Checking of protection information transferred from the data-out buffer	None	None	Based on the VRPROTECT field (see table 69)	
Comparing of transferred protection information	None	None	Perform a byte-by-byte comparison of protection information read from the medium and transferred from the data-out buffer based on the VRPROTECT field (see table 70)	

(a) – Medium verification of referenced unmapped LBAs shall be considered good.
(b) – Referenced unmapped LBAs retrieve protection information of FFFF_FFFF_FFFF_FFFFh (see 4.4.1.3), which disables protection checking.
(c) – The device server shall terminate the command with CHECK CONDITION status with the sense key set to MISCOMPARE and the additional sense code set to MISCOMPARE VERIFY OF UNMAPPED LBA (1D/xx)

The order of the user data and protection information checks and comparisons is vendor-specific.

If a byte-by-byte comparison is unsuccessful for any reason, then the device server shall terminate the command with CHECK CONDITION status with the sense key set to MISCOMPARE and the additional sense code set to the appropriate value for the condition.

The VERIFICATION LENGTH field specifies the number of contiguous logical blocks that shall be verified, starting with the logical block specified by the LOGICAL BLOCK ADDRESS field. If the BYTCHK bit is set to one, then the VERIFICATION LENGTH field also specifies the number of logical blocks that the device server shall transfer from the data-out buffer. A VERIFICATION LENGTH field set to zero specifies that no logical blocks shall be verified. This condition shall not be considered as an error. Any other value specifies the number of logical blocks that shall be verified. If the LBA plus the verification length exceeds the capacity of the medium, then the device server shall terminate the command with CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to LOGICAL BLOCK ADDRESS OUT OF RANGE. The VERIFICATION LENGTH field is constrained by the MAXIMUM TRANSFER LENGTH field in the Block Limits VPD page (see 6.4.2).

<…>

6.3 Mode parameters

6.3.1 Mode parameters overview

This subclause defines the block descriptors and mode pages used with direct-access block devices.

The mode parameter list, including the mode parameter header, is described in SPC-4. Direct-access block devices support zero or one mode parameter block descriptors (i.e., the block descriptor is shared by all the logical blocks on the medium).

The MEDIUM TYPE field in the mode parameter header (see SPC-4) shall be set to 00h.

The DEVICE-SPECIFIC PARAMETER field in the mode parameter header (see SPC-4) is defined for direct-access block devices in table 119.

Table 119 — DEVICE-SPECIFIC PARAMETER field for direct-access block devices

Bit	7	6	5	4	3	2	1	0
	WP	Reserved		DPOFUA	Reserved			

When used with the MODE SELECT command, the write protect (WP) bit is not defined.

When used with the MODE SENSE command, a WP bit set to one indicates that the medium is write-protected. A WP bit set to zero indicates that the medium is not write-protected. When the software write protect (SWP) bit in the Control mode page (see SPC-4) is set to one, the WP bit shall be set to one. When the SWP bit in the Control mode page is set to zero, the WP bit shall be set to one if the medium is write-protected (e.g., due to mechanisms outside the scope of this standard) or zero if the medium is not write-protected.

When used with the MODE SELECT command, the DPOFUA bit is reserved.

When used with the MODE SENSE command, a DPOFUA bit set to zero indicates that the device server does not support the DPO and FUA bits. When used with the MODE SENSE command, a DPOFUA bit set to one indicates that the device server supports the DPO and FUA bits (see 4.11).

<…>

6.4 Vital product data (VPD) parameters

6.4.1 VPD parameters overview

<…>

6.4.2 Block Limits VPD page

The Block Limits VPD page (see table 132) provides the application client with the means to obtain certain operating parameters of the logical unit.

Table 132 — Block Limits VPD page

Byte \ Bit	7	6	5	4	3	2	1	0
0	PERIPHERAL QUALIFIER			PERIPHERAL DEVICE TYPE				
1	PAGE CODE (B0h)							
2	Reserved							
3	PAGE LENGTH (18h)							
4	Reserved							
5								
6	(MSB)		OPTIMAL TRANSFER LENGTH GRANULARITY					
7								(LSB)
8	(MSB)		MAXIMUM TRANSFER LENGTH					
11								(LSB)
12	(MSB)		OPTIMAL TRANSFER LENGTH					
15								(LSB)
16	(MSB)		MAXIMUM PREFETCH XDREAD XDWRITE TRANSFER LENGTH					
19								(LSB)
20	(MSB)		MAXIMUM UNMAP LBA COUNT					
23								(LSB)
24	(MSB)		MAXIMUM UNMAP DESCRIPTOR COUNT					
27								(LSB)

The PERIPHERAL QUALIFIER field and the PERIPHERAL DEVICE TYPE field are defined in SPC-4.

The PAGE CODE field and PAGE LENGTH field are defined in SPC-4 and shall be set to the values defined in table 132.

The OPTIMAL TRANSFER LENGTH GRANULARITY field indicates the optimal transfer length granularity in blocks for a single ORWRITE command, PRE-FETCH command, READ command, VERIFY command, WRITE command, WRITE AND VERIFY command, XDREAD command, XDWRITE command, XDWRITEREAD command, or XPWRITE command. Transfers with transfer lengths not equal to a multiple of this value may incur significant delays in processing.

The MAXIMUM TRANSFER LENGTH field indicates the maximum transfer length in blocks that the device server accepts for a single ORWRITE command, READ command, VERIFY command, WRITE command, WRITE AND VERIFY command, XDWRITEREAD command, or XPWRITE command. Requests for transfer lengths exceeding this limit result in CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB. A MAXIMUM TRANSFER LENGTH field set to zero indicates that there is no reported limit on the transfer length.

The OPTIMAL TRANSFER LENGTH field indicates the optimal transfer length in blocks for a single ORWRITE command, PRE-FETCH command, READ command, VERIFY command, WRITE command, WRITE AND VERIFY command, XDREAD command, XDWRITE command, XDWRITEREAD command, or XPWRITE command. Transfers with transfer lengths exceeding this value may incur significant delays in processing.

The MAXIMUM PREFETCH XDREAD XDWRITE TRANSFER LENGTH field indicates:

a) the maximum transfer length in blocks that the device server accepts for a single PRE-FETCH command;
b) if the XOR Control mode page (see 6.3.7) is implemented, then the maximum value supported by the MAXIMUM XOR WRITE SIZE field in the XOR Control mode page; and
c) if the XOR Control mode page is not implemented, then the maximum transfer length in blocks that the device server accepts for a single XDWRITE command or XDREAD command.

The device server should set the MAXIMUM PREFETCH XDREAD XDWRITE TRANSFER LENGTH field to less than or equal to the MAXIMUM TRANSFER LENGTH field.

The MAXIMUM UNMAP LBA COUNT field indicates the maximum number of LBAs that may be represented in the parameter data transferred to the device server in the UNMAP PARAMETER LIST (see 5.x.2). If there is no limit on the number of LBAs that may be represented, then the device server shall set the MAXIMUM UNMAP LBA COUNT field to FFFF_FFFFh. If the logical unit implements thin provisioning, then this value shall be greater than or equal to one. A value of zero indicates that the logical unit does not support thin provisioning (see 4.4.1.5).

The MAXIMUM UNMAP DESCRIPTOR COUNT field indicates the maximum number of UNMAP DESCRIPTORs (see 5.x.2). If there is no limit on the number of UNMAP DESCRIPTORs that may be transferred, then the device server shall set the MAXIMUM UNMAP DESCRIPTOR COUNT field to FFFF_FFFFh. If the logical unit implements thin provisioning, then this value shall be greater than or equal to one. A value of zero indicates that the logical unit does not support thin provisioning (see 4.4.1.5).

SPC4 Changes

<...>

7.2.13 Statistics and Performance log pages
7.2.13.1 Statistics and Performance log pages overview

The Statistics and Performance log pages consist of a General Statistics and Performance log page and up to 31 Group Statistics and Performance log pages. Each Group Statistics and Performance log pages only collects statistics and performance information for the group number specified in a read CDB or a write CDB.

The General Statistics and Performance log page (see 7.2.13.2) provides the following statistics and performance results associated to the addressed logical unit:

a) Number of read commands;
b) Number of write commands;
c) Number of read logical blocks transmitted by a target port;
d) Number of write logical blocks received by a target port;
e) Read command processing time;
f) Write command processing time;
g) Sum of the command weights of the read commands plus write commands;
h) Sum of the weighted command time of the read commands plus write commands;
i) Idle time; and
j) Time interval.

The Group Statistics and Performance log pages (see 7.2.13.3) provide the following statistics and performance results associated to the addressed logical unit and the GROUP NUMBER field:

a) Number of read commands;
b) Number of write commands;
c) Number of read logical blocks transmitted by a target port;
d) Number of write logical blocks received by a target port;
e) Read command processing time; and
f) Write command processing time.

In the Statistics and Performance log pages, read and write commands are those shown in table 309.

Table 309 – Statistics and Performance log pages read and write commands

Read commands (a)	Write commands(a)

READ(6)
READ(10)
READ(12)
READ(16)
READ(32)
	UNMAP
	WRITE(6)
	WRITE(10)
	WRITE(12)
	WRITE(16)
	WRITE(32)
	WRITE AND VERIFY (10)
	WRITE AND VERIFY (12)
	WRITE AND VERIFY (16)
	WRITE AND VERIFY (32)

(a) See SBC-3.	