

To: T10 Technical Committee
From: Rob Elliott, HP (elliott@hp.com)
Date: 3 March 2006
Subject: 06-034r1 SBC-3 Physical blocks

Revision history

Revision 0 (4 January 2006) First revision
Revision 1 (3 March 2006) Incorporated comments from January 2006 CAP WG.

Related documents

sbc3r04 - SCSI Block Commands - 3 (SBC-3) revision 4
sat-r07 - SCSI to ATA Translation (SAT) revision 7
INCITS 317-1998 ATA Attachment -4 with Packet Interface Extension (ATA/ATAPI-4)
T13/d1699r02 - AT Attachment - 8 ATA/ATAPI Command Set (ATA8-ACS) revision 2

Related web sites

Big Sector consortium - <http://www.bigsector.org> (Maxtor, Seagate, Western Digital, Hitachi, Intel, LSI Logic, Microsoft)
IDEMA Symposium: HDD Dynamics--Interfaces, Electronics, Architecture and Reliability 6 December 2005: Session 3: HDD Sector Architecture. Slides at:
http://www.idema.org/_smartsite/modules/news/show_news.php?cmd=display&news_id=1230.
IDEMA Committee on Long Data Block Standards. Various presentations at:
http://www.idema.org/_smartsite/modules/local/data_file/show_file.php?cmd=standards&cat=103

Overview

ATA devices are starting to support physical sectors that are larger than logical sectors (see Related web sites). This improves error correction capability, increases capacity, and increases data rates.

The SCSI to ATA Translation (SAT) standard defines how to map an ATA device into a SCSI logical unit, but SCSI currently defines no way to report the ATA device's physical sector size or logical sector alignment.

ATA/ATAPI-7 and ATA8-ACS define a *Long Physical Sector feature set* (see 4.21) which allows an ATA device to present a 512 byte logical sector interface for ATA media access commands while implementing behind-the-scenes a larger physical sector size. The feature set lets the physical sector size be a 2ⁿ multiple of the logical sector size - 1, 2, 4, 8, ... 32,768 logical blocks corresponding to 512, 1024, 2048, 4096, ... 16,777,216 bytes per physical block.

If a read command accesses less than the physical sector size, the ATA device takes a bit longer to read extra data (probably saving it into a read cache). If a write command accesses less than the physical sector size, the ATA device performs a read-modify-write, which does have a noticeable performance impact. IDENTIFY DEVICE data Word 106 contains the number of logical sectors per physical sector (2ⁿ where n is 0 through 15). In addition to SAT concerns, native SCSI disk drives may want to implement this feature.

ATA8-ACS also allows logical sectors to not be aligned to the physical sectors (e.g., the ATA device could be designed such that a 1024 byte access at LBA 1 is aligned but LBA 0 is not). This was added because Master Boot Record (MBR) partitioned disks generally contain a single partition starting at LBA 63 (not the well-aligned LBA 64). Thus, when using a disk with 4096 byte physical sectors, performance will be better if LBAs 7/15/23/31/39/47/55/63/... are aligned to the physical sector boundary rather than LBAs 0/8/16/24/32/40/48/56/64/.... IDENTIFY DEVICE data Word 209 defines the offset of LBA 0 within a physical sector (14 bits, to allow up to LBA 16,383). It's not clear why this doesn't support the maximum value supported by Word 106 (which would take 15 bits). In addition to SAT concerns, native SCSI disk drives may want to implement this feature.

ATA/ATAPI-7 and ATA8-ACS also support logical sectors that are not 512 bytes. Words 117-118 contain the logical sector size in 16-bit words (with word 106 bit 12 also set to 1). SCSI already provides the equivalent functionality in the READ CAPACITY BLOCK LENGTH IN BYTES field and the mode parameter block descriptor BLOCK LENGTH field.

This proposal suggests these changes to SBC-3:

- a) Add a LOGICAL BLOCKS PER PHYSICAL BLOCK field to the READ CAPACITY (16) data to indicate the number of logical blocks per physical block.
- b) Add a LOGICAL BLOCK OFFSET field to the READ CAPACITY (16) data to indicate the offset of the logical blocks within the physical blocks. The field size is the same as the field size of the ATA IDENTIFY DEVICE command so SAT can fully translate an ATA device.
- c) Add a LOGICAL BLOCK OFFSET field to FORMAT UNIT to specify the alignment of LBA 0 within a physical block (default of 0 means naturally aligned). The field size of 6 bits only supports alignment up to LBA 63 (i.e., 512 byte logical block length to 32,768 byte physical block length, which should allow plenty of time for the need for this feature to dissipate).
- d) Add a PBLOCK bit to READ LONG and WRITE LONG to access a physical block rather than a logical block. For legacy software compatibility, the default accesses just a logical block. This requires a device server supporting large physical blocks to be able to mark individual logical blocks as containing errors and to use a format for the long data designed that a WRITE LONG to one logical block has no affect on any other logical blocks. If it cannot do so, it must terminate the command.
- e) Add a WR_UNCOR bit to WRITE LONG to mark the specified logical or physical block as containing an error. A SATL can directly translate this into the ATA8-ACS WRITE UNCORRECTABLE EXT command, which was added by proposal e02126r6 and induces an error on either a specific physical block (called a "pseudo-uncorrectable error") or a specific logical block (called a "flagged error"). ATA8-ACS does not define a WRITE LONG command, so it is not possible to translate SCSI WRITE LONG into anything guaranteed to have the same effect in ATA.

Editor's Note 1: This feature is particularly important for copy utilities that need to replicate bad logical blocks on disks, or RAID controllers that need to mark an error on a failed regenerated logical block. If legacy software is using SCSI WRITE LONG to do this, however, SAT cannot determine whether it is trying to cause an error or not. It would help if ATA resurrected its WRITE LONG command (last defined in ATA/ATAPI-4) and added a WRITE LONG EXT to map directly to the SCSI versions. Otherwise, SCSI software will have to change to use this new bit

- f) Allow FORMAT UNIT to treat all logical blocks sharing a physical block with a specified LBA in a defect list as defective. Since the entire disk is being formatted, this does not break legacy compatibility (as it would for REASSIGN BLOCKS or WRITE LONG).

Suggested changes

0.0.1 direct-access block device: A device that is capable of containing data stored in [logical](#) blocks that each have a unique logical block address.

0.0.2 logical block: A set of data bytes accessed and referenced as a unit [by the application client](#). See 4.4.

0.0.3 logical block address (LBA): The value used to reference a logical block.

0.0.4 physical block: [A set of data bytes accessed as a unit by the device server](#). See 4.x.

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-3, SPC-3, SCC-2, SES-2, and SMC-2.

Direct-access block devices store data for later retrieval in [logical/physical](#) blocks, [which are divided into logical blocks for most media-access commands](#). [Logical/Physical](#) 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). [Logical blocks contain user data and may contain](#)

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

Each logical block is stored at a unique logical block address (LBA), which is either four bytes (i.e., a short LBA) or eight bytes (i.e., a long LBA) in length. The logical block addresses on a logical block shall begin with zero and shall be contiguous up to the last logical block on the logical unit. 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 causes one or more logical blocks to be read from the medium. A verify operation confirms that one or more logical blocks were correctly written and are able to be read without error from the medium.

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.

4.2.2 Rotating media

The typical application of a direct-access block device is a magnetic disk device. The medium is a spinning disk with a magnetic material that allows flux changes to be induced and recorded. An actuator positions a read-write head radially across the spinning disk, allowing the device to randomly read or write the information at any radial position. Data is stored by using the write portion of the head to record flux changes and is read by using the read portion of the head to read the recorded data.

The circular path followed by the read-write head at a particular radius is called a track. The track is divided into sectors each containing blocks of stored data. If there are more than one disk spinning on a single axis and the actuator has one or more read-write heads to access the disk surfaces, the collection of tracks at a particular radius is called a cylinder.

A logical block is stored in one or more sectors, or a sector may store more than one logical block. Sectors may also contain information for accessing, synchronizing, and protecting the integrity of the logical blocks.

A rotating media-based direct-access block device is ready when the disks are rotating at the correct speed and the read-write circuitry is powered and ready to access the data, and may require a START STOP UNIT command (see 5.17) to bring the logical unit to the ready state.

Rotating media-based direct-access block device are usually non-volatile.

The defect management scheme of a disk device may not be discernible through this command set, though some aspects (see) may be accessible to the application client with the READ LONG commands and the WRITE LONG commands (see 5.14, 5.15, 5.33, and 5.34).

4.2.3 Memory media

Memory media is based on solid state random access memories (RAMs) (e.g., static RAM (SRAM), dynamic RAM (DRAM), magnetoresistive RAM (MRAM), ferroelectric RAM (FeRAM), or flash memory). Memory media-based direct-access block devices may be used for fast-access storage.

A memory media-based direct-access block device is ready after power on, and does not require a START STOP UNIT command (see 5.17) to bring the logical unit to a ready state.

These logical units may be non-mechanical, and therefore logical blocks may be accessed with similar access times regardless of their location on the medium. Memory media-based direct-access block devices may store less data than disks or tapes, and may be volatile.

The defect management scheme (e.g., ECC bytes) (see) may be accessible to the application client with the READ LONG commands and the WRITE LONG commands (see 5.14, 5.15, 5.33, and 5.34).

Memory media may be volatile (e.g., SRAM or DRAM) or non-volatile (e.g., SRAM or DRAM with battery backup, MRAM, FeRAM, or flash memory).

4.4 Logical blocks

Logical blocks are stored on the medium along with additional information that the device server uses to manage the 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, 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, and to provide protection against the loss of user data and protection information, if any (e.g., by containing ECC bytes).

The first logical block address is zero. The last logical block address is $[n-1]$, where $[n]$ is the number of logical blocks on the medium accessible by the application client. A READ CAPACITY command should be used to determine the value of $[n-1]$.

Logical block addresses 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)). The READ CAPACITY (10) command returns a capacity of FFFFFFFFh if the capacity exceeds that accessible with short LBAs, indicating that:

- a) the application client should enable descriptor format sense data (see SPC-3) in the Control mode page (see SPC-3) and in any REQUEST SENSE commands (see SPC-3) 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 1 - If a command with a 4-byte LOGICAL BLOCK ADDRESS field accesses logical blocks beyond logical block address FFFFFFFFh and fixed format sense data is used, there is no field in the sense data large enough to report the logical block address of an error (see 4.13).

If a command is received that references or attempts to access a logical block not within the capacity of the medium, 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 command may be terminated 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 block length. The parameter data returned by the READ CAPACITY command (see 5.10) describes the block length that is used on the medium. The mode parameter block descriptor (see 6.3.2) is used to change the block length in direct-access block devices that support changeable block lengths. The block length does not include the length of protection information and additional information, if any.

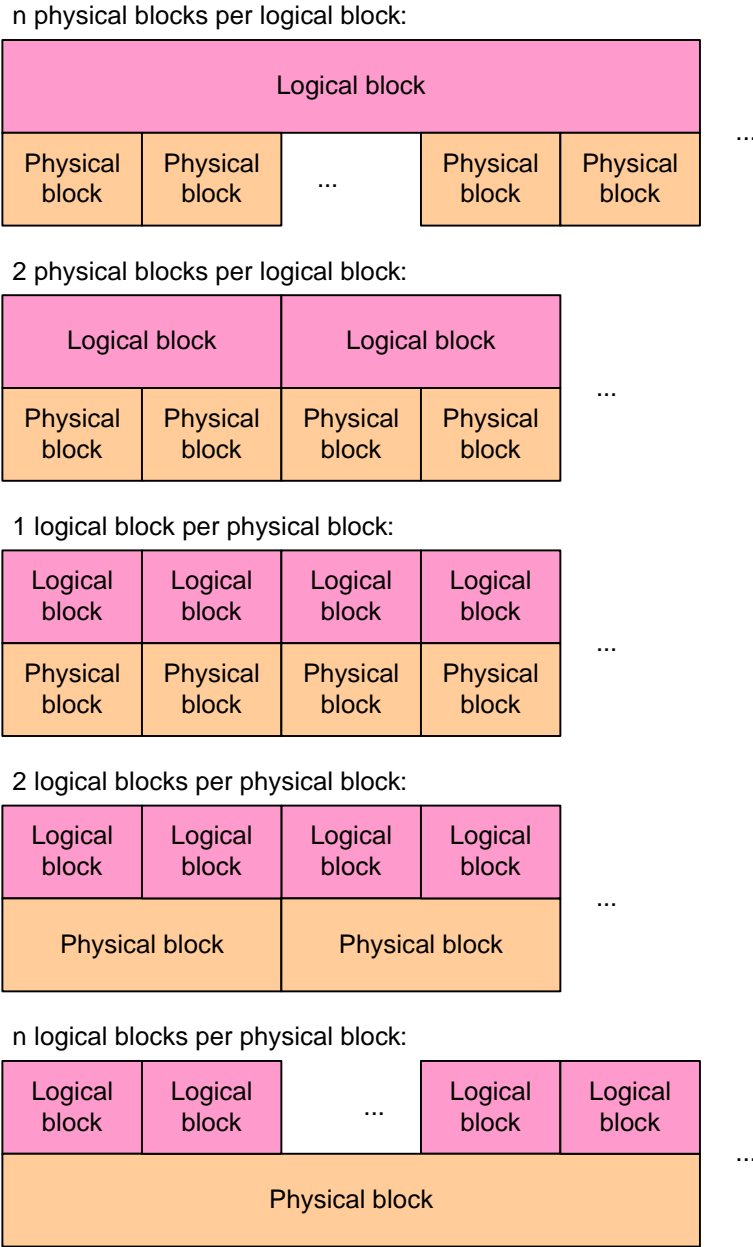
The location of a logical block on the medium is not required to have a relationship to the location of any other logical block. However, in a typical direct-access block device, the time to access a logical block at address $[x+1]$ after accessing logical block $[x]$ is often less than the time to access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

4.x Physical blocks

A physical block is a set of data bytes on the medium that are accessed by the device server as a unit. A physical block may contain a portion of a logical block, a complete logical block, or more than one logical blocks. Each physical block includes additional information that the device server uses to manage storage and retrieval. The device server shall be able to mark individual logical blocks as containing uncorrectable errors.

Direct-access block devices may be formatted into physical blocks that are larger than the logical block length (e.g., a physical block length of 4 096 bytes with a logical block length of 512 bytes) to increase performance, increase the efficiency of the ECC, and increase capacity.

[Figure 1 shows examples of logical blocks and physical blocks.](#)



[Figure 1 — Logical blocks and physical blocks examples](#)

[Logical blocks may or may not be aligned to the physical block boundaries.](#)

[Figure 2 shows examples of logical block offsets.](#)

2 logical blocks per physical blocks, logical block offset of 0 (natural alignment):

Logical block 0	Logical block 1	Logical block 2	Logical block 3	Logical block 4	Logical block 5	Logical block 6	Logical block 7	...
Physical block		Physical block		Physical block		Physical block		...

2 logical blocks per physical blocks, logical block offset of 1:

Unused	Logical block 0	Logical block 1	Logical block 2	Logical block 3	Logical block 4	Logical block 5	Logical block 6	...
Physical block		Physical block		Physical block		Physical block		...

4 logical blocks per physical blocks, logical block offset of 0 (natural alignment):

Logical block 0	Logical block 1	Logical block 2	Logical block 3	Logical block 4	Logical block 5	Logical block 6	Logical block 7	...
Physical block				Physical block				...

4 logical blocks per physical blocks, logical block offset of 1:

Unused	Logical block 0	Logical block 1	Logical block 2	Logical block 3	Logical block 4	Logical block 5	Logical block 6	...
Physical block				Physical block				...

4 logical blocks per physical blocks, logical block offset of 2:

Unused	Logical block 0	Logical block 1	Logical block 2	Logical block 3	Logical block 4	Logical block 5	...	
Physical block				Physical block				...

4 logical blocks per physical blocks, logical block offset of 3 (e.g. ideal for a disk with 512 byte logical blocks and 2048 byte physical blocks containing an MBR partition at LBA 63):

Unused	Logical block 0	Logical block 1	Logical block 2	Logical block 3	Logical block 4	...		
Physical block				Physical block				...

[Figure 2 — Logical block offset examples](#)

[To avoid incurring a performance penalty on media access commands, application clients should:](#)

- a) [specify a logical block that is aligned to a physical block boundary \(e.g., set the LOGICAL_BLOCK_ADDRESS field to an integral multiple of the logical block offset indicated by the LOGICAL_BLOCK_OFFSET field in the READ CAPACITY \(16\) parameter data \(see 5.11.2\)\); and](#)

- b) [access an integral number of physical blocks \(e.g., set the TRANSFER LENGTH field to an integral multiple of the number of logical blocks indicated by the LOGICAL BLOCKS PER PHYSICAL BLOCK field in the READ CAPACITY \(16\) parameter data \(see 5.11.2\)\).](#)

4.6 Initialization

Direct-access block devices may require initialization prior to write, read, and verify operations. This initialization is performed by a FORMAT UNIT command (see 5.2). Parameters related to the format (e.g., block **size****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, or verify operations. Mode parameters may also need initialization after logical unit resets.

NOTE 2 - Mode parameter block descriptors read with the MODE SENSE command before a FORMAT UNIT completes return 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 returned by the READ CAPACITY (10) command (see 5.10) or the READ CAPACITY (16) command (see 5.11) changes (e.g., when a FORMAT UNIT command or a MODE SELECT command completes changing the number of **logical** blocks, **logical** block **size****length**, protection information, or reference tag ownership values, or when a vendor-specific mechanism causes a change), the device server should establish a unit attention condition for the initiator port associated with each I_T nexus except the I_T nexus on which the command causing the change was received with an additional sense code of CAPACITY DATA HAS CHANGED.

NOTE 3 - Logical units compliant with previous versions of this standard did not establish a unit attention condition.

4.8 Medium defects

Any medium has the potential for defects that cause data to be lost. Therefore, each logical block may contain additional information that allows the detection of changes to the user data and protection information, if any, caused by defects in the medium or other phenomena, and may also allow the data to be reconstructed following the detection of such a change (e.g., ECC bytes). Some direct-access block devices allow the application client to examine and modify the additional information by using the READ LONG commands and the WRITE LONG commands (see 5.14, 5.15, 5.33, and 5.34). The application client may use the WRITE LONG commands to induce a defect to test the defect detection logic of the direct-access block device or to emulate an unrecoverable logical block when generating a mirror copy.

Defects may also be detected and managed during processing of the FORMAT UNIT command (see 5.2). The FORMAT UNIT command defines four sources of defect information: the PLIST, CLIST, DLIST, and GLIST. These defects may be reassigned or avoided during the initialization process so that they do not affect any logical blocks. The sources of defect location information (i.e., defects) are defined as follows:

- a) Primary defect list (PLIST). This is the list of defects, which may be supplied by the original manufacturer of the device or medium, that are considered permanent defects. The PLIST is located outside of the application client accessible logical block space. The PLIST is accessible by the device server for reference during the format operation, but it is not accessible by the application client except through the READ DEFECT DATA commands (see 5.10 and 5.13). Once created, the original PLIST shall not change;

- b) Logical unit certification list (CLIST). This list includes defects detected by the device server during an optional certification process performed during the FORMAT UNIT command. This list shall be added to the GLIST;
- c) Data defect list (DLIST). This list of defects may be supplied by the application client to the device server during the FORMAT UNIT command. This list shall be added to the GLIST; and
- d) Grown defect list (GLIST). The GLIST includes all defects sent by the application client (i.e., the DLIST) or detected by the device server (i.e., the CLIST). The GLIST does not include the PLIST. If the CMLST bit is set to zero, the GLIST shall include DLISTs provided to the device server during the previous and the current FORMAT UNIT commands. The GLIST shall also include:
 - A) defects detected by the format operation during medium certification;
 - B) defects previously identified with a REASSIGN BLOCKS command (see 5.16); and
 - C) defects previously detected by the device server and automatically reallocated.

The direct-access block device may automatically reassign defects if allowed by the Read-Write Error Recovery mode page (see 6.3.4).

Defects may also occur after initialization. The application client issues a REASSIGN BLOCKS command (see 5.16) to request that the specified **logical block address** **LBA** be reassigned to a different part of the medium. This operation may be repeated if a new defect appears at a later time. The total number of defects that may be handled in this manner is vendor-specific.

Defect management on direct-access block devices is vendor-specific. Direct-access block devices not using a removable medium may optimize the defect management for capacity or performance or both. Some direct-access block devices that use a removable medium do not support defect management or use defect management that does not impede the ability to interchange the medium.

5.2 FORMAT UNIT command

5.2.1 FORMAT UNIT command overview

The FORMAT UNIT command (see table 1) requests that the device server format the medium into application client accessible logical blocks as specified in the number of blocks and block length values received in the last mode parameter block descriptor (see 6.3.2) in a MODE SELECT command (see SPC-3). 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 the device server shall use the number of blocks and block length at which the logical unit is currently formatted (i.e., no change is made to the number of blocks and the block length of the logical unit during the format operation).

Table 1 — FORMAT UNIT command

Byte/Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (04h)							
1	FMPINFO	RTO_REQ	LONGLIST	FMTDATA	CMPLIST	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 respond to commands attempting to enter into the task set 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. Handling of commands already in the task set is vendor-specific.

The PROGRESS INDICATION field in parameter data returned in response to a REQUEST SENSE command (see SPC-3) 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-3).

A format protection information (FMTPINFO) bit set to zero specifies that the device server shall disable the use of protection information (see 4.16) and format the medium to the block length specified in the mode parameter block descriptor of the mode parameter header (see SPC-3). A FMTPINFO bit set to one specifies that the device server shall enable the use of protection information (see 4.16) and format the medium to the block length specified in the mode parameter block descriptor of the mode parameter header plus eight (e.g., if the block length is 512, then the formatted block length is 520). Following a successful format, the PROT_EN bit in the READ CAPACITY (16) parameter data (see 5.11) indicates whether protection information (see 4.16) is enabled.

The reference tag own request (RTO_REQ) bit specifies whether the application client or the device server has ownership of the LOGICAL BLOCK REFERENCE TAG field in protection information (see 4.16.2). If the FMTPINFO bit is set to zero and the RTO_REQ bit is set to one, 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 CDB. If the FMTPINFO bit is set to one and the RTO_REQ bit is set to one, the device server shall enable application client ownership of the LOGICAL BLOCK REFERENCE TAG field. If the FMTPINFO bit set to one and the RTO_REQ bit is set to zero, the device server shall disable application client ownership (i.e., enable device server ownership) of the LOGICAL BLOCK REFERENCE TAG field. Following a successful format, the RTO_EN bit in the READ CAPACITY (16) parameter data (see 5.11) indicates whether application client ownership of the LOGICAL BLOCK REFERENCE TAG field is enabled.

When protection information is written during a FORMAT UNIT command (i.e., the FMTPINFO bit is set to one) protection information shall be written to a default value of FFFFFFFF_FFFFFFFFh.

A LONGLIST bit set to zero specifies that the parameter list, if any, contains a short parameter list header as defined in table 4. A LONGLIST bit set to one specifies that the parameter list, if any, contains a long parameter list header as defined in table 5. If the FMTPINFO bit is set to zero, the LONGLIST bit shall be ignored.

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

A FMTPDATA bit set to one specifies that the FORMAT UNIT parameter list (see table 3) 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.

A complete list (CMPLST) bit set to zero specifies that the defect list included in the FORMAT UNIT parameter list shall be used in an addition to the existing list of defects. As a result, the device server shall construct a new GLIST (see 4.8) that contains:

- a) the existing GLIST;
- b) the DLIST, if it is sent by the application client; and
- c) the CLIST, if certification is enabled (i.e., the device server may add any defects it detects during the format operation).

A Cmplst bit set to one specifies that the defect list included in the FORMAT UNIT parameter list is a complete list of defects. Any existing defect list except the PLIST shall be ignored by the device server. As a result, the device server shall construct a new GLIST (see 4.8) that contains:

- a) the DLIST, if it is sent by the application client; and
- b) the CLIST, if certification is enabled (i.e., the device server may add any defects it detects during the format operation).

If the FMTDATA bit is set to zero, the Cmplst bit shall be ignored.

The DEFECT LIST FORMAT field specifies the format of the address descriptors in the defect list if the FMTDATA bit is set to one (see table 2).

Table 2 defines the address descriptor usage for the FORMAT UNIT command.

Table 2 — FORMAT UNIT command address descriptor usage

Field in the FORMAT UNIT CDB			DEFECT LIST LENGTH field in the parameter list header	Type ^a	Comments ^f
FMTDATA	Cmplst	DEFECT LIST FORMAT			
0	any	000b	Not available	M	Vendor-specific defect information
1	0	000b (short block)	Zero	O	See ^b and ^d
1	1			O	See ^b and ^e
1	0		Nonzero	O	See ^c and ^d
1	1			O	See ^b and ^e
		011b (long block)	Zero	O	See ^b and ^d
				O	See ^b and ^e
1	0		Nonzero	O	See ^c and ^d
1	1			O	See ^c and ^e

^a M = implementation is mandatory. O = implementation is optional.

^b No DLIST is included in the parameter list.

^c A DLIST is included in the parameter list. The device server shall add the DLIST defects to the new GLIST.

^d The device server shall add existing GLIST defects to the new GLIST (i.e., use the existing GLIST).

^e The device server shall not add existing GLIST defects to the new GLIST (i.e., discard the existing GLIST).

^f All the options described in this table cause a new GLIST to be created during processing of the FORMAT UNIT command as described in the text.

Table 2 — FORMAT UNIT command address descriptor usage

Field in the FORMAT UNIT CDB			DEFECT LIST LENGTH field in the parameter list header	Type ^a	Comments ^f
FMTDATA	CMPLST	DEFECT LIST FORMAT			
1	0	100b (bytes from index)	Zero	O	See ^b and ^d
1	1			O	See ^b and ^e
1	0		Nonzero	O	See ^c and ^d
1	1			O	See ^c and ^e
1	0	101b (physical sector)	Zero	O	See ^b and ^d
1	1			O	See ^b and ^e
1	0		Nonzero	O	See ^c and ^d
1	1			O	See ^c and ^e
1	0	110b (vendor specific)	Vendor specific	O	
1	1			O	
All others				Reserved.	

^a M = implementation is mandatory. O = implementation is optional.
^b No DLIST is included in the parameter list.
^c A DLIST is included in the parameter list. The device server shall add the DLIST defects to the new GLIST.
^d The device server shall add existing GLIST defects to the new GLIST (i.e., use the existing GLIST).
^e The device server shall not add existing GLIST defects to the new GLIST (i.e., discard the existing GLIST).
^f All the options described in this table cause a new GLIST to be created during processing of the FORMAT UNIT command as described in the text.

5.2.2 FORMAT UNIT parameter list

5.2.2.1 FORMAT UNIT parameter list overview

Table 3 defines the FORMAT UNIT parameter list.

Table 3 — FORMAT UNIT parameter list

Byte\Bit	7	6	5	4	3	2	1	0
0 to 3 or 0 to 7	Parameter list header (see table 4 or table 5 in 5.2.2.2)							
	Initialization pattern descriptor (if any)(see table 7 in 5.2.2.3)							
	Defect list (if any)							

The parameter list header is defined in 5.2.2.2.

The initialization pattern descriptor, if any, is defined in 5.2.2.3.

The defect list, if any, contains address descriptors (see 5.2.2.4) each specifying a location on the medium that the device server shall exclude from the application client accessible part. This is called the DLIST (see 4.8)

5.2.2.2 Parameter list header

The parameter list headers (see table 4 and table 5) provide several optional format control parameters. Device servers that implement these headers provide the application client additional control over the use of the four defect sources, and the format operation. If the application client attempts to select any function not implemented by the device server, 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.

The short parameter list header (see table 4) is used if the LONGLIST bit is set to zero in the FORMAT UNIT CDB.

Table 4 — Short parameter list header

Byte\Bit	7	6	5	4	3	2	1	0
0	Reserved					PROTECTION FIELDS USAGE		
1	FOV	DPRY	DCRT	STPF	IP	Obsolete	IMMED	Vendor specific
2	(MSB) _____							
3	DEFECT LIST LENGTH							(LSB)

The long parameter list header (see table 5) is used if the LONGLIST bit is set to one in the FORMAT UNIT CDB.

Table 5 — Long parameter list header

Byte\Bit	7	6	5	4	3	2	1	0
0	Reserved					PROTECTION FIELDS USAGE		
1	FOV	DPRY	DCRT	STPF	IP	Obsolete	IMMED	Vendor specific
2	Reserved							
3	Reserved		LOGICAL BLOCK OFFSET					
4	(MSB) _____							
7	DEFECT LIST LENGTH							(LSB)

A format options valid (FOV) bit set to zero specifies that the device server shall use its default settings for the DPRY, DCRT, STPF, and IP bits. If the FOV bit is set to zero, the application client shall set these bits to zero. If the FOV bit is set to zero and any of the other bits listed in this paragraph are not set to zero, 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 FOV bit set to one specifies that the device server shall examine the values of the DPRY, DCRT, STPF, and IP bits. When the FOV bit is set to one, the DPRY, DCRT, STPF, and IP bits are defined as follows.

A disable primary (DPRY) bit set to zero specifies that the device server shall not use parts of the medium identified as defective in the PLIST for application client accessible logical blocks. If the device server is not able to locate the PLIST or it is not able to determine whether a PLIST exists, it shall take the action specified by the STPF bit.

A D_{PRY} bit set to one specifies that the device server shall not use the PLIST to identify defective areas of the medium. The PLIST shall not be deleted.

A disable certification (DCRT) bit set to zero specifies that the device server shall perform a vendor-specific medium certification operation to generate a CLIST. A DCRT bit set to one specifies that the device server shall not perform any vendor-specific medium certification process or format verification operation.

The stop format (STPF) bit controls the behavior of the device server if one of the following events occurs:

- a) The device server has been requested to use the PLIST (i.e., the D_{PRY} bit is set to zero) or the GLIST (i.e., the C_{MPLST} bit is set to zero) and the device server is not able to locate the list or determine whether the list exists; or
- b) The device server has been requested to use the PLIST (i.e., the D_{PRY} bit is set to zero) or the GLIST (i.e., the C_{MPLST} bit is set to zero), and the device server encounters an error while accessing the defect list.

A STPF bit set to zero specifies that, if one or both of these events occurs, the device server shall continue to process the FORMAT UNIT command. The device server shall return CHECK CONDITION status at the completion of the FORMAT UNIT command with the sense key set to RECOVERED ERROR and the additional sense code set to either DEFECT LIST NOT FOUND if the condition described in item a) occurred, or DEFECT LIST ERROR if the condition described in item b) occurred.

A STPF bit set to one specifies that, if one or both of these events occurs, the device server shall terminate the FORMAT UNIT command with CHECK CONDITION status and the sense key shall be set to MEDIUM ERROR with the additional sense code set to either DEFECT LIST NOT FOUND if the condition described in item a) occurred, or DEFECT LIST ERROR if the condition described in item b) occurred.

NOTE 4 - The use of the F_MT_DA_TA bit, the C_MP_LS_T bit, and the parameter list header allow the application client to control the source of the defect lists used by the FORMAT UNIT command. Setting the DEFECT LIST LENGTH field to zero allows the application client to control the use of PLIST and CLIST without having to specify a DLIST.

An initialization pattern (IP) bit set to zero specifies that an initialization pattern descriptor is not included and that the device server shall use its default initialization pattern. An IP bit set to one specifies that an initialization pattern descriptor (see 5.2.2.3) is included in the FORMAT UNIT parameter list following the parameter list header.

An immediate (IMMED) bit set to zero specifies that the device server shall return status after the format operation has completed. An IMMED bit value set to one specifies that the device server shall return status after the entire parameter list has been transferred.

The DEFECT LIST LENGTH field specifies the total length in bytes of the defect list (i.e., the address descriptors) that follows and does not include the initialization pattern descriptor, if any. The formats for the address descriptor(s) are shown in 5.2.2.4.

[The LOGICAL BLOCK OFFSET field specifies the logical block offset that the device server shall establish and is defined in table 6.](#)

Table 6 — LOGICAL BLOCK OFFSET field

Code	Description
0	Vendor-specific
m > 0	Each logical block with an LBA that is an integral multiple of (m-1) shall be located at the beginning of a physical block

Short block format address descriptors and long block format address descriptors should be in ascending order. Bytes from index format address descriptors and physical sector format address descriptors shall be in ascending order. More than one physical or logical block may be affected by each address descriptor. If the address descriptors are not in the required order, 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.

5.2.2.3 Initialization pattern descriptor

The initialization pattern descriptor specifies that the device server initialize logical blocks to a specified pattern. The initialization pattern descriptor (see table 7) is sent to the device server as part of the FORMAT UNIT parameter list.

Table 7 — Initialization pattern descriptor

Byte\Bit	7	6	5	4	3	2	1	0
0	IP MODIFIER		SI	Reserved				
1	INITIALIZATION PATTERN TYPE							
2	(MSB)	INITIALIZATION PATTERN LENGTH (n - 3)						(LSB)
3								
4	INITIALIZATION PATTERN							
n								

The initialization pattern modifier (IP MODIFIER) field (see table 8) specifies the type and location of a header that modifies the initialization pattern.

Table 8 — Initialization pattern modifier (IP MODIFIER) field

Code	Description
00b	No header. The device server shall not modify the initialization pattern.
01b	The device server shall overwrite the initialization pattern to write the LBA in the first four bytes of each logical block. The LBA shall be written with the most significant byte first. If the LBA is larger than four bytes, the least significant four bytes shall be written ending with the least significant byte.
10b	The device server shall overwrite the initialization pattern to write the LBA in the first four bytes of each physical block contained within the logical block. The lowest numbered logical block or part thereof that occurs within the physical block is used. The LBA shall be written with the most significant byte first. If the LBA is larger than four bytes the least significant four bytes shall be written ending with the least significant byte.
11b	Reserved.

A security initialize (SI) bit set to one specifies that the device server shall attempt to write the initialization pattern to all areas of the medium including those that may have been reassigned (i.e., are in a defect list). An SI bit set to one shall take precedence over any other FORMAT UNIT CDB field. The initialization pattern shall be written using a security erasure write technique. Application clients may choose to use this command multiple times to fully erase the previous data. Such security erasure write technique procedures are outside the scope of this standard. The exact requirements placed on the security erasure write technique are vendor-specific. The intent of the security erasure write is to render any previous user data unrecoverable by any analog or digital technique.

An SI bit set to zero specifies that the device server shall initialize the application client accessible part of the medium. The device server is not required to initialize other areas of the medium. However, the device server shall format the medium as defined in the FORMAT UNIT command.

When the SI bit is set to one, the device server need not write the initialization pattern over the header and other header and other parts of the medium not previously accessible to the application client. If the device

server is unable to write over any part of the medium that is currently accessible to the application client or may be made accessible to the application client in the future (e.g., by clearing the defect list), it shall terminate the command with CHECK CONDITION status with the sense key set to MEDIUM ERROR and the additional sense code set to the appropriate value for the condition. The device server shall attempt to rewrite all remaining parts of the medium even if some parts are not able to be rewritten.

The INITIALIZATION PATTERN TYPE field (see table 9) specifies the type of pattern the device server shall use to initialize each logical block within the application client accessible part of the medium. All bytes within a logical block shall be written with the initialization pattern. The initialization pattern is modified by the IP MODIFIER field as described in table 8.

Table 9 — INITIALIZATION PATTERN TYPE field

Code	Description
00h	Use a default initialization pattern ^a
01h	Repeat the pattern specified in the INITIALIZATION PATTERN field as required to fill the logical block ^b
02h - 7Fh	Reserved
80h - FFh	Vendor-specific
^a If the INITIALIZATION PATTERN LENGTH field is not set to zero, 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. ^b If the INITIALIZATION PATTERN LENGTH field is set to zero, 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.	

The INITIALIZATION PATTERN LENGTH field specifies the number of bytes contained in the INITIALIZATION PATTERN field. If the initialization pattern length exceeds the current block length 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.

The INITIALIZATION PATTERN field specifies the initialization pattern. The initialization pattern is modified by the IP MODIFIER field.

5.2.2.4 Address descriptor formats

5.2.2.4.1 Address descriptor formats overview

This subclause describes the address descriptor formats used in the FORMAT UNIT command, the READ DEFECT DATA commands (see 5.12 and 5.13), and the Translate Address diagnostic pages (see 6.1.2 and 6.1.3) of the SEND DIAGNOSTIC command and the RECEIVE DIAGNOSTIC RESULTS command.

The format type of an address descriptor is specified with:

- a) the DEFECT LIST FORMAT field in the CDB, for the FORMAT UNIT command and the READ DEFECT DATA commands;
- b) the SUPPLIED FORMAT field, for the Translate Address diagnostic pages; or
- c) the TRANSLATE FORMAT field, for the Translate Address diagnostic pages.

Table 10 defines the types of address descriptors.

Table 10 — Address descriptor formats

Format type	Description	Reference
000b	Short block format address descriptor	5.2.2.4.2
011b	Long block format address descriptor	5.2.2.4.3
100b	Bytes from index format address descriptor	5.2.2.4.4
101b	Physical sector format address descriptor	5.2.2.4.5
110b	Vendor-specific	
All others	Reserved	

5.2.2.4.2 Short block format address descriptor

A format type of 000b specifies the short block format address descriptor defined in table 11.

Table 11 — Short block format address descriptor (000b)

Byte\Bit	7	6	5	4	3	2	1	0
0	(MSB) _____ SHORT BLOCK ADDRESS _____ (LSB)							
3								

For the FORMAT UNIT command, the SHORT BLOCK ADDRESS field contains the four-byte LBA of a defect. [The device server may consider all the other logical blocks in the same physical block as the specified logical block as containing defects.](#)

Editor’s Note 2: This is not dangerous (like with REASSIGN BLOCKS and WRITE LONG) since the whole disk is being formatted.

For the READ DEFECT DATA commands, the SHORT BLOCK ADDRESS field contains a vendor-specific four-byte value.

For the Translate Address diagnostic pages, the SHORT BLOCK ADDRESS field contains a four-byte LBA or a vendor-specific four-byte value that is greater than the capacity of the medium.

Editor’s Note 3: Above broken into 3 paragraphs

5.2.2.4.3 Long block format address descriptor

A format type of 011b specifies the long block format address descriptor defined in table 12.

Table 12 — Long block format address descriptor (011b)

Byte\Bit	7	6	5	4	3	2	1	0
0	(MSB) _____ LONG BLOCK ADDRESS _____ (LSB)							
7								

For the FORMAT UNIT command, the LONG BLOCK ADDRESS field contains the eight-byte ~~logical block address~~LBA of a defect. The device server may consider all the other logical blocks in the same physical block as the specified logical block as containing defects.

Editor's Note 4: This is not dangerous (like with REASSIGN BLOCKS and WRITE LONG) since the whole disk is being formatted.

For the READ DEFECT DATA commands, the LONG BLOCK ADDRESS field contains a vendor-specific eight-byte value.

For the Translate Address diagnostic pages, the LONG BLOCK ADDRESS field contains a ~~four~~eight-byte LBA or a vendor-specific ~~four~~eight-byte value that is greater than the capacity of the medium.

Editor's Note 5: Above broken into 3 paragraphs

5.2.2.4.4 Bytes from index format address descriptor

A format type of 100b specifies the bytes from index address descriptor defined in table 13. For the FORMAT UNIT command and the READ DEFECT DATA commands, this descriptor specifies the location of a defect that is either the length of one track or is no more than eight bytes long. For the Translate Address diagnostic pages, this descriptor specifies the location of a track or the first byte or last byte of an area.

Table 13 — Bytes from index format address descriptor (100b)

Byte\Bit	7	6	5	4	3	2	1	0
0	(MSB)							
2	CYLINDER NUMBER							(LSB)
3	HEAD NUMBER							
4	(MSB)							
7	BYTES FROM INDEX							(LSB)

The CYLINDER NUMBER field contains the cylinder number.

The HEAD NUMBER field contains the head number.

The BYTES FROM INDEX field contains the number of bytes from the index (e.g., from the start of the track) to the location being described. A BYTES FROM INDEX field set to FFFFFFFFh specifies that the entire track is being described.

For sorting bytes from index format address descriptors, the cylinder number is the most significant part of the address and the bytes from index is the least significant part of the address. More than one logical block may be described by this descriptor.

5.2.2.4.5 Physical sector format address descriptor

A format type of 101b specifies the physical sector address descriptor defined in table 14. For the FORMAT UNIT command and the READ DEFECT DATA commands, this descriptor specifies the location of a defect

that is either the length of one track or the length of one sector. For the Translate Address diagnostic pages, this descriptor specifies the location of a track or a sector.

Table 14 — Physical sector format address descriptor (101b)

Byte\Bit	7	6	5	4	3	2	1	0	
0	(MSB)								
	CYLINDER NUMBER								
2								(LSB)	
3	HEAD NUMBER								
4	(MSB)								
	SECTOR NUMBER								
7								(LSB)	

The CYLINDER NUMBER field contains the cylinder number.

The HEAD NUMBER field contains the head number.

The SECTOR NUMBER field contains the sector number. A SECTOR NUMBER field set to FFFFFFFFh specifies that the entire track is being described.

For sorting physical sector format address descriptors, the cylinder number is the most significant part of the address and the sector number is the least significant part of the address. More than one logical block may be described by this descriptor.

5.11 READ CAPACITY (16) command

...

5.11.2 READ CAPACITY (16) parameter data

The READ CAPACITY (16) parameter data is defined in table 15. Any time the READ CAPACITY (16) parameter data changes, the device server should establish a unit attention condition as described in 4.6.

Table 15 — READ CAPACITY (16) parameter data

Byte\Bit	7	6	5	4	3	2	1	0	
0	(MSB)								
	RETURNED LOGICAL BLOCK ADDRESS								
7								(LSB)	
8	(MSB)								
	BLOCK LENGTH IN BYTES								
11								(LSB)	
12	Reserved						RTO_EN	PROT_EN	
13	Reserved				LOGICAL BLOCKS PER PHYSICAL BLOCK				
14	Reserved		(MSB)	LOGICAL BLOCK OFFSET					
15								(LSB)	
16	Reserved								
31	Reserved								

The RETURNED LOGICAL BLOCK ADDRESS field and 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 5.10). The maximum value that shall be returned in the RETURNED LOGICAL BLOCK ADDRESS field is FFFFFFFF_FFFFFFFEh.

A reference tag own enable (RTO_EN) bit set to one indicates that application client ownership of the LOGICAL BLOCK REFERENCE TAG field in protection information is enabled (i.e., the medium was formatted with

protection information (see 4.16) enabled and the RTO_REQ bit was set to one). An RTO_EN bit set to zero indicates that application client ownership of the LOGICAL BLOCK REFERENCE TAG field in protection information is disabled.

A PROT_EN bit set to one indicates that the medium was formatted with protection information (see 4.16) enabled. A PROT_EN bit set to zero indicates that the medium was not formatted with protection information enabled.

The LOGICAL BLOCKS PER PHYSICAL BLOCK field is defined in table 16.

Table 16 — LOGICAL BLOCKS PER PHYSICAL BLOCK field

<u>Code</u>	<u>Description</u>
<u>0</u>	<u>No information provided</u>
<u>1</u>	<u>1 logical block per physical block, or more than one physical block per logical block</u>
<u>n > 1</u>	<u>2⁽ⁿ⁻¹⁾ logical blocks per physical block</u>

The LOGICAL BLOCK OFFSET field is defined in table 17.

Table 17 — LOGICAL BLOCKS OFFSET field

<u>Code</u>	<u>Description</u>
<u>0</u>	<u>No information provided</u>
<u>m > 0</u>	<u>Each logical block with an LBA that is an integral multiple of (m-1) is located at the beginning of a physical block</u>

5.3 PRE-FETCH

...

The LOGICAL BLOCK ADDRESS field specifies the first logical block accessed by this command. If the logical block address exceeds the capacity of the medium 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.

Editor’s Note 6: above paragraph included to show that the standard LOGICAL BLOCK ADDRESS field definition is not adequate for READ LONG/WRITE LONG

5.14 READ LONG (10) command

The READ LONG (10) command (see table 18) requests that the device server transfer data from a single logical or physical block to the data-in buffer. The data transferred during the READ LONG (10) command is vendor-specific, but shall include the following items recorded on the medium:

- a) if a physical block is being transferred:
 - A) user data or transformed user data for all the logical blocks in the physical block;
 - B) protection information or transformed protection information, if any, for all the logical blocks in the physical block; and
 - C) any additional information (e.g., ECC bytes);

and
- b) if a logical block is being transferred:
 - A) user data or transformed user data for the logical block;
 - B) protection information or transformed protection information, if any, for the logical block; and
 - C) any additional information (e.g., ECC bytes) for all the physical blocks in the logical block.

If a cache contains a more recent version of [the specified](#) logical [or physical](#) block, the device server shall write the logical [or physical](#) block to the medium before reading it. The values in the Read-Write Error Recovery mode page (see 6.3.4) do not apply to this command. The device server may perform retries while processing this command.

Table 18 — READ LONG (10) command

Byte\Bit	7	6	5	4	3	2	1	0	
0	OPERATION CODE (3Eh)								
1	Reserved					PBLOCK	CORRCT	Obsolete	
2	(MSB)	LOGICAL BLOCK ADDRESS						(LSB)	
5									
6	Reserved								
7	(MSB)	BYTE TRANSFER LENGTH						(LSB)	
8									
9	CONTROL								

~~See the PRE-FETCH (10) command (see 5.3) for the definition of the LOGICAL BLOCK ADDRESS field.~~

The LOGICAL BLOCK ADDRESS field specifies a logical block. If the logical block address exceeds the capacity of the medium 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 additional information contain an ECC, any other additional bytes that are correctable by ECC should be included (e.g., a data synchronization mark within the area covered by ECC). It is not required for the ECC bytes to be at the end of the user data or protection information, if any; however, the ECC bytes should be in the same order as they are on the medium.

A PBLOCK bit set to one specifies that the device server shall return the entire physical block containing the specified logical block. A PBLOCK bit set to zero specifies that the device server shall return bytes representing only the specified logical block.

A correct (CORRCT) bit set to zero specifies that a logical block be read without any correction made by the device server. A CORRCT bit set to one should result in GOOD status unless data is not transferred for some reason other than that the data is non-correctable. In this case the appropriate status and sense data shall be returned. A CORRCT bit set to one specifies that the data be corrected by ECC before being transferred to the data-in buffer.

The BYTE TRANSFER LENGTH field specifies the number of bytes of data that shall be read from the specified logical [or physical](#) block and transferred to the data-in buffer. If the BYTE TRANSFER LENGTH field is not set to zero and does not match the available data length, 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 CDB. In the sense data (see 4.13 and SPC-3), the VALID and ILI bits shall each be set to one and the INFORMATION field shall be set to the difference (i.e., residue) of the requested byte transfer length minus the actual available data length in bytes. Negative values shall be indicated by two's complement notation.

A BYTE TRANSFER LENGTH field set to zero specifies that no bytes shall be read. This condition shall not be considered an error.

5.15 READ LONG (16) command

The READ LONG (16) command (see table 19) requests that the device server transfer data from a single logical or physical block to the data-in buffer. The data transferred during the READ LONG (16) command is vendor-specific, but shall include the following items recorded on the medium:

- a) if a physical block is being transferred:
 - A) user data or transformed user data for all the logical blocks in the physical block;
 - B) protection information or transformed protection information, if any, for all the logical blocks in the physical block; and
 - C) any additional information (e.g., ECC bytes);-

and

- b) if a logical block is being transferred:
 - A) user data or transformed user data for the logical block;
 - B) protection information or transformed protection information, if any, for the logical block; and
 - C) any additional information (e.g., ECC bytes) for all the physical blocks in the logical block.

If a cache contains a more recent version of the specified logical or physical block, the device server shall write the logical or physical block to the medium before reading it. The values in the Read-Write Error Recovery mode page (see 6.3.4) do not apply to this command. The device server may perform retries while processing this command. This command is implemented as a service action of the SERVICE ACTION IN operation code (see A.2).

Table 19 — READ LONG (16) command

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (9Eh)							
1	Reserved			SERVICE ACTION (11h)				
2	(MSB)	LOGICAL BLOCK ADDRESS						(LSB)
9								
10	Reserved							
11								
12	(MSB)	BYTE TRANSFER LENGTH						(LSB)
13								
14	Reserved						PBLOCK	CORRCT
15	CONTROL							

See the READ LONG (10) command (see) for the definitions of the fields in this command.

5.16 REASSIGN BLOCKS command

5.16.1 REASSIGN BLOCKS command overview

The REASSIGN BLOCKS command (see table 20) 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 4.8).

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, it shall write the recovered user data and any protection information to the reassigned logical block. If the device server is unable to recover user data and protection information, if any, it shall write vendor-specific data as the user data and shall write a default value of FFFFFFFF_FFFFFFFFh as the protection information, if enabled. The data in all other logical blocks on the medium shall be preserved.

NOTE 5 - The effect of specifying a logical block to be reassigned that previously has been reassigned is to reassign the logical block again. Although not likely, over the life of the medium, a logical block may be assigned to multiple physical block addresses until no more spare locations remain on the medium.

Table 20 — REASSIGN BLOCKS command

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (07h)							
1	Reserved						LONGLBA	LONGLIST
2	Reserved							
4	Reserved							
5	CONTROL							

A long LBA (LONGLBA) bit set to zero specifies that the REASSIGN BLOCKS defective LBA list contains four byte LBAs. A LONGLBA bit set to one specifies that the REASSIGN BLOCKS defective LBA list contains eight byte LBAs.

5.16.2 REASSIGN BLOCKS parameter list

The REASSIGN BLOCKS parameter list (see table 21) contains a four-byte parameter list header followed by a defective LBA list containing one or more LBAs.

Table 21 — REASSIGN BLOCKS parameter list

Byte\Bit	7	6	5	4	3	2	1	0
0	Parameter list header (see table 22 or table 23)							
3	Parameter list header (see table 22 or table 23)							
4	DEFFECTIVE LBA LIST (if any)							
n	DEFFECTIVE LBA LIST (if any)							

If LONGLIST is set to zero, the parameter list header is defined in table 22.

Table 22 — REASSIGN BLOCKS short parameter list header

Byte\Bit	7	6	5	4	3	2	1	0	
0	Reserved								
1	Reserved								
2	(MSB)	DEFECT LIST LENGTH							
3							(LSB)		

If LONGLIST is set to one, the parameter list header is defined in table 23.

Table 23 — REASSIGN BLOCKS long parameter list header

Byte\Bit	7	6	5	4	3	2	1	0	
0	(MSB)	DEFECT LIST LENGTH							
3							(LSB)		

The DEFECT LIST LENGTH field indicates the total length in bytes of the DEFECTIVE LBA LIST field. The DEFECT LIST LENGTH field does not include the parameter list header length and is equal to either:

- a) four times the number of LBAs, if the LONGLBA bit is set to zero; or
- b) eight times the number of LBAs, if the LONGLBA bit is set to one.

The DEFECTIVE LBA LIST field contains a list of defective LBAs. Each LBA is a four-byte field if the LONGLBA bit is set to zero or an eight-byte field if the LONGLBA bit is set to one. The LBAs shall be in ascending order.

If the direct-access block device has insufficient capacity to reassign all of the specified logical blocks, the device server shall terminate the command with CHECK CONDITION status with the sense key set to HARDWARE ERROR and the additional sense code set to NO DEFECT SPARE LOCATION AVAILABLE.

If the direct-access block device is unable to successfully complete a REASSIGN BLOCKS command, the device server shall terminate the command with CHECK CONDITION status with the appropriate sense data (see 4.13 and SPC-3). The first LBA not reassigned shall be returned in the COMMAND-SPECIFIC INFORMATION field of the sense data. If information about the first LBA not reassigned is not available, or if all the defects have been reassigned, the COMMAND-SPECIFIC INFORMATION field shall be set to FFFFFFFFh if fixed format sense data is being used or FFFFFFFF_FFFFFFFFh if descriptor format sense data is being used.

If the REASSIGN BLOCKS command failed due to an unexpected unrecoverable read error that would cause the loss of data in a logical block not specified in the defective LBA list, the LBA of the unrecoverable block shall be returned in the INFORMATION field of the sense data and the VALID bit shall be set to one.

NOTE 6 - If the REASSIGN BLOCKS command returns CHECK CONDITION status and the sense data COMMAND-SPECIFIC INFORMATION field contains a valid LBA, the application client should remove all LBAs from the defective LBA list prior to the one returned in the COMMAND-SPECIFIC INFORMATION field. If the sense key is MEDIUM ERROR and the INFORMATION field contains the valid LBA, the application client should insert that new defective LBA into the defective LBA list and reissue the REASSIGN BLOCKS command with the new defective LBA list. Otherwise, the application client should perform any corrective action indicated by the sense data and then reissue the REASSIGN BLOCKS command with the new defective LBA list.

5.33 WRITE LONG (10) command

The WRITE LONG (10) command (see table 24) requests that the device server [mark a logical or physical block as containing an error, or](#) transfer data for a single logical [or physical](#) block from the data-out buffer and write it to the medium. The data written shall be the same length and shall be in the same order as the data returned by the READ LONG (10) command (see). The device server shall write the logical [or physical](#) block to the medium, and shall not return GOOD status until the logical block has actually been written on the medium.

Table 24 — WRITE LONG (10) command

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (3Fh)							
1	COR_DIS	WR_UNCOR	PBLOCK	Reserved				Obsolete
2	(MSB) LOGICAL BLOCK ADDRESS (LSB)							
5	Reserved							
6	Reserved							
7	(MSB) BYTE TRANSFER LENGTH (LSB)							
8	Reserved							
9	CONTROL							

~~See the PRE-FETCH (10) command (see 5.3) for the definition of the LOGICAL BLOCK ADDRESS field.~~

A correction disabled (COR_DIS) bit set to zero specifies that, when the specified logical block is read, the device server shall perform normal error recovery on that logical block. A COR_DIS bit set to one specifies that, when the specified logical block is read, the device server shall:

- a) perform no error recovery on that logical block including any read error recovery enabled by the Read-Write Error Recovery mode page (see 6.3.5);
- b) perform no automatic reallocation of that logical block including any automatic reallocation enabled by the Read-Write Error Recovery mode page;
- c) not consider errors on logical blocks to be informational exception conditions as defined in the Information Exceptions Control mode page (see SPC-4); and
- d) return CHECK CONDITION status with the sense key set to MEDIUM ERROR and the additional sense code set to READ ERROR - LBA MARKED BAD BY APPLICATION CLIENT.

The condition established by the COR_DIS bit being set to one shall remain in effect until the logical block is written by any means (e.g., any WRITE command, WRITE SAME command, FORMAT command, or another WRITE LONG command specifying the same logical block with the COR_DIS bit set to zero).

A WR_UNCOR bit set to one specifies that the device server shall mark the specified logical or physical block as containing an uncorrectable error, shall ignore the BYTE TRANSFER LENGTH field, and shall transfer no data. A WR_UNCOR bit set to zero specifies nothing.

A PBLOCK bit set to one specifies that the device server shall mark or write the entire physical block containing the specified logical block. A PBLOCK bit set to zero specifies that the device server shall mark or write only the specified logical block.

The LOGICAL BLOCK ADDRESS field specifies a logical block. If the logical block address exceeds the capacity of the medium 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 BYTE TRANSFER LENGTH field specifies the number of bytes of data that the device server shall transfer from the data-out buffer and write to the specified logical or physical block, provided that the WR_UNCOR bit is set to zero. If the BYTE TRANSFER LENGTH field is not set to zero and does not match the data length that the device server returns for a READ LONG command, 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 CDB. In the sense data (see 4.13 and SPC-3), the ILI and VALID bits shall be set to one and the INFORMATION field shall be set to the difference (i.e., residue) of the requested length minus the actual length in bytes. Negative values shall be indicated by two's complement notation. A BYTE TRANSFER LENGTH field set to zero specifies that no bytes shall be written. This condition shall not be considered an error.

5.34 WRITE LONG (16) command

The WRITE LONG (16) command (see table 25) requests that the device server mark a logical or physical block as containing an error, or transfer data for a single logical or physical block from the data-out buffer and write it to the medium. The data written shall be the same length and shall be in the same order as the data returned by the READ LONG (16) command (see). The device server shall write the logical or physical block to the medium, and shall not return GOOD status until the logical block has actually been written on the medium. This command is implemented as a service action of the SERVICE ACTION OUT operation code (see A.2).

Table 25 — WRITE LONG (16) command

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (9Fh)							
1	COR_DIS	WR_UNCOR	PBLOCK	SERVICE ACTION (11h)				
2	(MSB)	LOGICAL BLOCK ADDRESS						(LSB)
9								
10	Reserved							
11								
12	(MSB)	BYTE TRANSFER LENGTH						(LSB)
13								
14	Reserved							
15	CONTROL							

See the WRITE LONG (10) command (see) for the definitions of the fields in this command.

5.35 WRITE SAME (10) command

...

Table 26 describes the LBDATA bit and the PBDATA bit.

Table 26 — LBDATA bit and PBDATA bit

LBDATA	PBDATA	Description
0	0	<p>The device server shall write the single block of user data received from the data-out buffer to each logical block without modification.</p> <p>If the medium is formatted with protection information:</p> <ul style="list-style-type: none"> a) the value in the LOGICAL BLOCK REFERENCE TAG field received in the single block of data from the data-out buffer shall be placed into the LOGICAL BLOCK REFERENCE TAG field of the first logical block written to the medium. Into each of the subsequent logical blocks, the device server shall place into the LOGICAL BLOCK REFERENCE TAG field the value of the previous logical block's LOGICAL BLOCK REFERENCE TAG field plus one; b) If the ATO bit is set to one in the Control mode page (see SPC-3), the logical block application tag received in the single block of data shall be placed in the LOGICAL BLOCK APPLICATION TAG field of each logical block. If the ATO bit is set to zero, the device server may write any value into the LOGICAL BLOCK APPLICATION TAG field of each logical block; and c) The value in the DATA BLOCK GUARD field received in the single block of data from the data-out buffer shall be placed in the DATA BLOCK GUARD field of each logical block.
0	1 ^a	The device server shall replace the first eight bytes of the block received from the data-out buffer to each physical sector with the physical address of the sector being written using the physical sector format (see 5.2.2.4.5).
1 ^a	0	The device server shall replace the first four bytes of the block received from the data-out buffer with the least significant four bytes of the LBA of the block being written, ending with the least significant byte (e.g., if the LBA is 77665544_33221100h, 33221100h is written with 33h written first and 00h written last).
1	1	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 CDB.
<p>^a If the medium is formatted with protection information then the protection information shall be written to a default value of FFFFFFFF_FFFFFFFFh in each of the written logical blocks.</p>		

5.2.3 6.3.2 Mode parameter block descriptors

5.2.3.1 Mode parameter block descriptors overview

If the device server returns a mode parameter block descriptor, it shall return a short LBA mode parameter block descriptor (see 5.2.3.2) in the mode parameter data in response to:

- a) a MODE SENSE (6) command; or
- b) a MODE SENSE (10) command with the LLBAA bit set to zero.

If the device server returns a mode parameter block descriptor and the number of [logical](#) blocks is greater than FFFFFFFFh, it may return a long LBA mode parameter block descriptor (see 5.2.3.3) in the mode parameter data in response to a MODE SENSE (10) command with the LLBAA bit set to one.

If the application client sends a mode parameter block descriptor in the mode parameter list, it shall send a short LBA mode parameter block descriptor (see 5.2.3.2) for a MODE SELECT (6) command.

If the application client sends a mode parameter block descriptor in the mode parameter list, it may send a long LBA mode parameter block descriptor (see 5.2.3.3) for a MODE SELECT (10) command.

Support for the mode parameter block descriptors is optional. The device server shall establish a unit attention condition with the additional sense code of MODE PARAMETERS CHANGED (see SPC-3 and SAM-3) when the block descriptor values are changed.

5.2.3.2 Short LBA mode parameter block descriptor

Table 27 defines the block descriptor for direct-access block devices used:

- a) with the MODE SELECT (6) and MODE SENSE (6) commands; and
- b) with the MODE SELECT (10) and MODE SENSE (10) commands when the LONGLBA bit is set to zero in the mode parameter header (see SPC-3).

Table 27 — Short LBA mode parameter block descriptor

Byte\Bit	7	6	5	4	3	2	1	0
0	(MSB) _____							
3	NUMBER OF BLOCKS							_____ (LSB)
4	Reserved							
5	(MSB) _____							
7	BLOCK LENGTH							_____ (LSB)

[Editor’s Note 7: Could rename these fields \(and all references to them\) to NUMBER OF LOGICAL BLOCKS and LOGICAL BLOCK LENGTH for consistency. Such changes are not marked in this revision, though.](#)

A device server shall respond to a MODE SENSE command (see SPC-3) by reporting the number of [logical](#) blocks specified in the NUMBER OF BLOCKS field sent in the last MODE SELECT command that contained a mode parameter block descriptor. If no MODE SELECT command with a mode parameter block descriptor has been received then the current number of [logical](#) blocks shall be returned. To determine the number of [logical](#) blocks at which the logical unit is currently formatted, the application client shall use the READ CAPACITY command (see 5.11) rather than the MODE SELECT command.

On a MODE SENSE command, the device server may return a value of zero indicating that it does not report the number of [logical](#) blocks in the short LBA mode parameter block descriptor.

On a MODE SENSE command, if the number of logical blocks on the medium exceeds the maximum value that is able to be specified in the NUMBER OF BLOCKS field, the device server shall return a value of FFFFFFFFh.

If the logical unit does not support changing its capacity by changing the NUMBER OF BLOCKS field using the MODE SELECT command (see SPC-3), the value in the NUMBER OF BLOCKS field is ignored. If the device supports changing its capacity by changing the NUMBER OF BLOCKS field, then the NUMBER OF BLOCKS field is interpreted as follows:

- a) If the NUMBER OF BLOCKS field is set to zero, the logical unit shall retain its current capacity if the block length has not changed. If the NUMBER OF BLOCKS field is set to zero and the content of the BLOCK LENGTH field (i.e., new block length) is different than the current block length, the logical unit shall be set to its maximum capacity when the new block length takes effect (i.e., after a successful FORMAT UNIT command);
- b) If the NUMBER OF BLOCKS field is greater than zero and less than or equal to its maximum capacity, the logical unit shall be set to that number of [logical](#) blocks. If the content of the BLOCK LENGTH field is the same as the current block length, the logical unit shall not become format corrupt. This capacity setting shall be retained through power cycles, hard resets, logical unit resets, and I_T nexus losses. If the content of the BLOCK LENGTH field is the same as the current block length this capacity setting shall take effect on successful completion of the MODE SELECT command. If the content of the

BLOCK LENGTH field (i.e., new block length) is different than the current block length this capacity setting shall take effect when the new block length takes effect (i.e., after a successful FORMAT UNIT command);

- c) If the NUMBER OF BLOCKS field is set to a value greater than the maximum capacity of the device and less than FFFFFFFFh, then the MODE SELECT 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 logical unit shall retain its previous block descriptor settings; or
- d) If the NUMBER OF BLOCKS field is set to FFFFFFFFh, the logical unit shall be set to its maximum capacity. If the content of the BLOCK LENGTH field is the same as the current block length, the logical unit shall not become format corrupt. This capacity setting shall be retained through power cycles, hard resets, logical unit resets, and I_T nexus losses. If the content of the BLOCK LENGTH field is the same as the current block length this capacity setting shall take effect on successful completion of the MODE SELECT command. If the content of the BLOCK LENGTH field (i.e., new block length) is different than the current block length this capacity setting shall take effect when the new block length takes effect (i.e., after a successful FORMAT UNIT command).

The BLOCK LENGTH field specifies the length in bytes of each logical block. No change shall be made to any logical blocks on the medium until a format operation (see 5.2) is initiated by an application client.

A device server shall respond to a MODE SENSE command (see SPC-3) by reporting the length of the logical blocks as specified in the BLOCK LENGTH field sent in the last MODE SELECT command that contained a mode parameter block descriptor. If no MODE SELECT command with a block descriptor has been received then the current block length shall be returned (e.g., if the block length is 512 bytes and a MODE SELECT command occurs with the BLOCK LENGTH field set to 520 bytes, any MODE SENSE commands would return 520 in the BLOCK LENGTH field). To determine the block length at which the logical unit is currently formatted, the application client shall use the READ CAPACITY command (see 5.11) rather than the MODE SELECT command.

5.2.3.3 Long LBA mode parameter block descriptor

Table 28 defines the block descriptor for direct-access block devices used with the MODE SELECT (10) command and MODE SENSE (10) command when the LONGLBA bit is set to one in the mode parameter header (see SPC-3).

Table 28 — Long LBA mode parameter block descriptor

Byte\Bit	7	6	5	4	3	2	1	0
0	(MSB)							
7	NUMBER OF BLOCKS							(LSB)
8	Reserved							
11	Reserved							
12	(MSB)							
15	BLOCK LENGTH							(LSB)

A device server shall respond to a MODE SENSE command (see SPC-3) by reporting the number of [logical](#) blocks specified in the NUMBER OF BLOCKS field sent in the last MODE SELECT command that contained a mode parameter block descriptor. If no MODE SELECT command with a mode parameter block descriptor has been received then the current number of [logical](#) blocks shall be returned. To determine the number of [logical](#) blocks at which the logical unit is currently formatted, the application client shall use the READ CAPACITY command (see 5.11) rather than the MODE SELECT command.

On a MODE SENSE command, the device server may return a value of zero indicating that it does not report the number of [logical](#) blocks in the long LBA mode parameter block descriptor.

If the logical unit does not support changing its capacity by changing the NUMBER OF BLOCKS field using the MODE SELECT command (see SPC-3), the value in the NUMBER OF BLOCKS field is ignored. If the device supports changing its capacity by changing the NUMBER OF BLOCKS field, then the NUMBER OF BLOCKS field is interpreted as follows:

- a) If the NUMBER OF BLOCKS field is set to zero, the logical unit shall retain its current capacity if the block length has not changed. If the NUMBER OF BLOCKS field is set to zero and the content of the BLOCK LENGTH field (i.e., new block length) is different than the current block length, the logical unit shall be set to its maximum capacity when the new block length takes effect (i.e., after a successful FORMAT UNIT command);
- b) If the NUMBER OF BLOCKS field is greater than zero and less than or equal to its maximum capacity, the logical unit shall be set to that number of [logical](#) blocks. If the content of the BLOCK LENGTH field is the same as the current block length, the logical unit shall not become format corrupt. This capacity setting shall be retained through power cycles, hard resets, logical unit resets, and I_T nexus losses. If the content of the BLOCK LENGTH field is the same as the current block length this capacity setting shall take effect on successful completion of the MODE SELECT command. If the content of the BLOCK LENGTH field (i.e., new block length) is different than the current block length this capacity setting shall take effect when the new block length takes effect (i.e., after a successful FORMAT UNIT command);
- c) If the NUMBER OF BLOCKS field is set to a value greater than the maximum capacity of the device and less than FFFFFFFF FFFFFFFFh, then the device server shall terminate the MODE SELECT 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. The logical unit shall retain its previous block descriptor settings; or
- d) If the NUMBER OF BLOCKS field is set to FFFFFFFF FFFFFFFFh, the logical unit shall be set to its maximum capacity. If the content of the BLOCK LENGTH field is the same as the current block length, the logical unit shall not become format corrupt. This capacity setting shall be retained through power cycles, hard resets, logical unit resets, and I_T nexus losses. If the content of the BLOCK LENGTH field is the same as the current block length this capacity setting shall take effect on successful completion of the MODE SELECT command. If the content of the BLOCK LENGTH field (i.e., new block length) is different than the current block length this capacity setting shall take effect when the new block length takes effect (i.e., after a successful FORMAT UNIT command).

The BLOCK LENGTH field specifies the length in bytes of each logical block. No change shall be made to any logical blocks on the medium until a format operation (see 5.2) is initiated by an application client.

A device server shall respond to a MODE SENSE command (see SPC-3) by reporting the length of the logical blocks as specified in the BLOCK LENGTH field sent in the last MODE SELECT command that contained a mode parameter block descriptor. If no MODE SELECT command with a block descriptor has been received then the current block length shall be returned (e.g., if the block length is 512 bytes and a MODE SELECT command occurs with the BLOCK LENGTH field set to 520 bytes, any MODE SENSE commands would return 520 in the BLOCK LENGTH field). To determine the block length at which the logical unit is currently formatted, the application client shall use the READ CAPACITY command (see 5.11) rather than the MODE SELECT command.

6.3.3 Caching mode page

...

The MINIMUM PRE-FETCH field specifies the number of [logical](#) blocks to pre-fetch regardless of the delays it might cause in processing subsequent commands. The field contains either:

- a) a number of [logical](#) blocks, if the MF bit is set to zero; or
- b) a scalar multiplier of the value in the TRANSFER LENGTH field, if the MF bit is set to one.

...

The MAXIMUM PRE-FETCH field specifies the number of [logical](#) blocks to pre-fetch if the pre-fetch does not delay processing of subsequent commands. The field contains either:

- a) a number of [logical](#) blocks, if the MF bit is set to zero; or
- b) a scalar multiplier of the value in the TRANSFER LENGTH field, if the MF bit is set to one.

The MAXIMUM PRE-FETCH field contains the maximum amount of data to pre-fetch as a result of one READ command. It is used in conjunction with the DISABLE PRE-FETCH TRANSFER LENGTH field and MAXIMUM PRE-FETCH CEILING field to trade off pre-fetching new data with displacing old data already stored in the cache.

The MAXIMUM PRE-FETCH CEILING field specifies an upper limit on the number of logical blocks computed as the maximum pre-fetch. If this number of [logical](#) blocks is greater than the value in the MAXIMUM PRE-FETCH field, then the number of logical blocks to pre-fetch shall be truncated to the value stored in the MAXIMUM PRE-FETCH CEILING field.