The Data Integrity Extension is an optional feature for direct access devices (peripheral device type 00). The Data Integrity Extensions are designed to provide end-to-end protection of user data against media and transmission errors. These extensions are carefully designed to allow for transparent usage in legacy contexts, i.e., the Data Integrity Extensions do not materially affect legacy operations (e.g., Read and Write).

The main control (the **STOR_DIF** flag) for the Data Integrity Extension is the controls in the new Mode page that indicate whether the Data Integrity Field (DIF) is appended to each block on the device. If these controls indicate that the Data Integrity Extensions are disabled (i.e., **STOR_DIF** is 0), the device operates in the normal legacy manner. If these controls indicate that the Data Integrity extensions are enabled (i.e. **STOR_DIF** is 1), each block on the device is expected to have a Data Integrity Field attached and the device is expected to process that DIF in the manner outlined in this document.

There are essentially five possible kinds of devices in the context of the Data Integrity Extensions, as follows:

?? The device does not implement the Data Integrity Mode page or any of the Data Integrity Extensions (a “legacy device” in this context).

?? The device implements the Data Integrity Mode page but forces the DIF flag to 0 (zero) and does not allow it to change. Such devices do not implement any other aspects of the Data Integrity Extensions and operate only as a legacy device. This form is called a “DIE aware” implementation.

?? The device implements DIFs internally but does not provide initiator access to the DIF fields. This form is called a “hidden DIF” implementation.

?? The device forces the attachment of a DIF to every data block. In this case the DIF flag (**STOR_DIF**) on the Data Integrity Mode page is forced to 1 and is marked as unchangeable. Because of the manner in which the Data Integrity Field is handled in normal Read/Write usages, such devices can be attached to an initiator that does not recognize the Data Integrity Extensions. This form is called a “forced DIF” implementation.

?? The device allows the DIF flag (**STOR_DIF**) in the Data Integrity Mode page to be altered and processes operations according to the setting of that flag. This form is called a “full DIE” implementation.

Moreover, a device may limit the scope of its implementation to a subset of the full Data Integrity functionality. In most cases, such limitations are indicated by limiting the accepted values for one or more of the Mode page control fields. The device may either reject any Mode Select command that attempts to set such a control to an unacceptable value, or it may further indicate that the value of a control field may not be changed. by omitting it from the “changeable values” form of the Mode page.

The remainder of this document describes the operational extensions and modifications associated with the Data Integrity Extensions.

The Data Integrity Field

The Data Integrity Field (DIF) is an 8-byte element with three sub-fields, as follows:[1]

4 bytes	2 bytes	2 bytes
Ref Tag	**Meta Tag**	**Guard**

The first sub-field (the Ref Tag) is a 4 byte value which is nominally intended to hold information which identifies the block within some context. The characteristics of this context are not described in this document. During a multi-block operation, this field is incremented (by 1) for each successive block processed. The normal/default content of this field is the low order 4 bytes of the Logical Block Address (LBA).

The second sub-field (the Meta Tag) is a 2 byte value, which is (normally) held fixed within the context of a single command. The contents of this field are generally only meaningful to an application and thus falls outside the scope of this document. It might, for example, hold information identifying a virtual LUN or flags indicating block state or status. This value is often provided by, or matches, the fixed value parameter in the Data Integrity Mode page.

The third sub-field is a 2 byte guard value computed from the user data block, nominally under control of the Data Integrity Mode page. The computation of this value does not include any tag parts of the DIF itself and may, as specified in the Data Integrity Mode page, exclude a trailing part of the user data block. The guard value is normally one of the following, with the method selected in the Data Integrity Mode page:

?? A modified 1's complement checksum[2] that can be easily calculated in code. The computed value can never be zero (0) and a guard value of 0 can be used to mark special block statuses (e.g., intentionally treat as a bad block).

?? A CRC. In this case, the guard value can be zero (0) and some other method to indicate a specially marked block is required. A specially marked block is identified by computing the nominal CRC value and modifying that value by XORing in a value of BADBh

The capability of specially marking data blocks is useful when an application (e.g., a RAID controller) may need to mark a block as bad (or special) while avoiding the possibility of spurious media errors (e.g., the old, standard, method of modifying ECC).

Format

When a device is formatted with the DIF flag active (i.e., the Data Integrity extensions enabled), the device will physically format the media with data blocks extended with a (default) DIF. These blocks will be 8 (or 16 if DIFs must be stacked) bytes (the size of the DIF) longer than the blocks that would be formatted if the DIF flag were not enabled. For example, if the device is set up to format 512 byte blocks if the DIF flag is disabled, enabling the DIF flag will cause the device to be formatted in 520 byte blocks.

---

[1] Future extensions may provide a 4 byte guard (or dual 2-byte guards) by removing the explicit Meta Tag field and possibly using the nominal Meta field value as a seed for the guard calculation.

[2] In RAID applications using the checksum guard method, the guard sub-field in a parity block is usually the XOR of the checksum guard values of the associated data blocks, protecting across the whole stripe..

As the result of a Format operation, each formatted block will consist of the following:

?? The user data will be the same as it would if the DIF flag was not enabled.
?? The REF Tag part of the DIF will be the low order 4 bytes of the LBA.
?? The META Tag part of the DIF will be as specified in the Data Integrity Mode page. This will be the META TAG DEFAULT value from the Mode page.
?? The guard sub-field will be calculated according to the settings in the Data Integrity Mode page.

Inquiry and Read Capacity

Bit 0 of byte 5 of the Inquiry data is defined to have the following meaning:

Bit 0 – Indicates that the device supports the Data Integrity Mode page.

The Read Capacity data returned by a device where the blocks have attached DIFs does not include the length of the DIF (8 bytes or possibly 16 bytes if DIFs must be “stacked”) in the value returned for the block size. For example, if the device is formatted for 512 byte user data blocks with attached DIFs, the physical block size on the media will be 520 bytes, but the block size returned by a Read Capacity command will be 512 bytes. Note that the number of data blocks is likely to be different depending on whether the device is formatted with attached DIFs or not.

Mode pages

The Data Integrity Extensions affect one existing Mode page (page 03, the “format” page) and add a new Data Integrity Mode page (page code TDB).

In the “format” mode page, bit 0 of byte 20 is defined to have the following meaning:

Bit 0 – The value of the **STOR_DIF** flag at the last format operation

This bit is informational only and should be marked as being unchangeable.

If the application client changes the DIF-enable flag (**STOR_DIF**), the **REF METHOD** value, the **GUARD METHOD** value, or the **META ECHO** value, the device may not be able to access the media until a FORMAT UNIT command has been successfully completed. *[Note to reviewers: See “Note 30” on page 114 of **sbc2r07**.pdf]*. Change in the latter three values may result in a “format required” state (see **STK_META**, **STK_REF**, and **STK_GRD**) when the “device” is really an intermediate controller which makes internal use of the tag fields, i.e., makes internal use of the META TAG field (e.g., to hold a virtual LUN number), requires a particular guard calculation method as part of a RAID algorithm, or requires LBA-locked REF TAGs to verify the accuracy of its internal mapping functions. In such cases, the “device” (intermediate controller) may “stack” DIFs by extending the user data block with two 8-byte DIFs with the first holding the “visible” DIF (and tag values) and the second holding the tag values used internally by the “device” (intermediate controller). The controller may either calculate the guard value in the second DIF over the user data block plus the first DIF or it may exclude the first DIF from the calculation (and exclude those 8 bytes from any back-end guard calculation). In the latter case, the guard value in the second DIF can generally be copied from the guard value present in the first DIF (assuming the same guard calculation method). A device which has specific requirements on the **REF METHOD**, **GUARD METHOD**, or the **META ECHO** controls may refuse to allow changes to those values rather than implementing some form of DIF “stacking” or some equivalent mechanism.

Detailed control over the Data Integrity Extensions is provided by the new Data Integrity Mode page (page code: TBD), which is laid out as follows:

Bit Byte	7	6	5	4	3	2	1	0
0	PS	RESERVED	PAGE CODE (TBDh)					
1	PAGE LENGTH (0Eh)							
2	STOR_DIF	Vendor Specific		LBA_ 40b	EXCL_Bytes (# 4B incr.)			
3	META ECHO	REF METHOD		GUARD METHOD			RESERVED	
4	{Primary Verification Controls:}		{Alternate Verification Controls:}		{Legacy Verification Controls:}		{Legacy Write Controls:}	
	REF_ CK_p	GRD_ CK_p	REF_ CK_a	GRD_ CK_a	REF_ CK_l	GRD_ CK_l	ZAP_ REF	MRK_ GRD
5	STK_ META	STK_ REF	STK_ GRD	DI_ AVAIL	Reserved			
6	(MSB) META TAG MASK Primary							
7	(LSB)							
8	(MSB) META TAG MASK Alternate							
9	(LSB)							
10	(MSB) META TAG MASK Legacy							
11	(LSB)							
12	(MSB) META TAG DEFAULT (legacy)							
13	(LSB)							
14								
15								

If the **STOR_DIF** flag is 0 (zero), the special DIF CDBs are not available and a Check condition with ILLEGAL OPERATION sense data should be issued in response. If the **STOR_DIF** flag is 1 (one), the special DIF CDBs are available and the device is normally expected to have been formatted with a DIF attached to each data block.

The **LBA_40b** flag is independent of the DIE proposal and is described in a separate document. It appears here for convenience only and might be migrated to a different, more appropriate, Mode page. This feature provides a larger LBA value in certain CDBs.

The **EXCL_Bytes** count, which must be 8 or less, indicates the number of 4 byte DWords, at the end of the data block, which are excluded from the guard calculation. In other words, the guard value is computed against the first (BlkSize-4***EXCL_Bytes**) bytes of the data block, where the BlkSize is the value reported in response to a Read Capacity command. For example, if the reported block size is 520 bytes and the **EXCL_Bytes** count is 2, the guard value will be calculated against the first 512 bytes of the block, while, if the reported block size is 512 bytes and the **EXCL_Bytes** count is 8, the guard value will be computed against the first 480 bytes of the data block. The data bytes excluded from the guard calculation in this way are sometimes called “Customer Specific Data”. Generally, the part of the block excluded from the guard calculation by the **EXCL_Bytes** count is called the “data block extension”, while the part of the block included in the guard calculation is usually called the “user data block”.

The controls in byte 3 of the Mode page describe the format and handling of the Tag and Guard sub-fields of the DIF. If a device only accepts one setting for a control, it should also mark that field as unchangeable.

The **META ECHO** flag indicates the behavior that the initiator observes with respect to the META tag sub-field. If the **META ECHO** flag is 0 (zero), the initiator expects (and requires) that the value returned in the META Tag field of any block read using a special DIF Read is the value last written using a DIF Write (or the value of the **META TAG DEFAULT** field at the time of the last legacy write). The returned value may or may not match (and cause a UNIT CHECK condition if the META Tag is being verified) the nominal Meta Tag value indicated in a DIF-Read CDB or expected during a normal Read. If the **META ECHO** flag is 1 (one), the initiator expects (and requires) that the device return the nominal Meta Tag value in a DIF Read in the META Tag sub-field of each returned DIF. On a DIF Write, when the META ECHO flag is 1, the device is expected to verify the META Tag sub-field in any received DIF, but is then, in effect, free to discard the value. Note that when the **META ECHO** flag is 1, the device must also, effectively, disable Meta Tag verification on Read operations.

The **REF METHOD** field controls the handling of, and any expectations regarding, the REF Tag sub-field. Note that the device may require that the initiator follow certain protocols with respect to the REF Tag sub-field. The meanings of these controls are:

REF METHOD	**REF Generation [Initial REF Tag value]**	**REF value returned for a DIF Read**
00	All Writes: LBA	LBA
01	Legacy Writes: LBA DIF Writes: From CDB	Value written by the last Write
10	Reserved (/ Vendor Specific)	
11		

Note that when the **REF METHOD** is **00b**, the device should reject special DIF operations if the REF value supplied in the CDB (when the CDB contains an explicit REF Tag value) differs from the above. In addition, when the **REF METHOD** is **00b**, the device is only required to verify the REF Tag value during a DIF-Write and to supply the appropriate REF Tag value in response to a DIF Read. An **REF METHOD** of **00b** is generally used (forced) by a device that is an intermediate controller. If a device cannot properly alter or supply the REF Tag value during a DIF Read, it should reject attempts to set the **REF METHOD** to **00b**.

The **GUARD METHOD** control specifies the method used to calculate the guard value from the data block. This calculated guard value is either used to generate the DIF (normal writes) or to verify the guard value sub-field in the DIF. A device is not required to support both methods. The GUARD METHOD control field provides a number of Reserved values to allow for future extensions, e.g., 4 byte guard calculations or a 4 byte guard consisting of two 2-byte (independently calculated) guard values.

The **GUARD METHOD** control has the following values:

GUARD SELECT	**Guard Calculation**
0 {000}	Reserved
1 {001}	The checksum-based algorithm, using 16-bit 1's complement addition with a left rotation (multiply by 2) each step
2 {010}	The CRC-based algorithm, with polynomial: {18BB7h or:} x16 + x15 + x11 + x9 + x8 + x7 + x5 + x4 + x2 + x + 1
3 {011} thru 7 {111}	Reserved / Vendor Specific

The checksum-based value can be easily calculated by code. The specific algorithms used to calculate this value is provided in an appendix (below). The CRC-based value is calculated by serially processing the bits of each byte into the polynomial from the high-order bit (bit 7) through the low-order bit (bit 0). The precise CRC calculation algorithm is described at the end of this document. Both calculations are seeded with a logical zero value (using the FFFFh form for the checksum to ensure that the calculated value is never 0000h) to make them easier to use in RAID applications.

The verification flags in byte 4 control the verification of the DIF during read, write, and verify operations. There are three sets of controls, with the following usages:

- ?? The primary set is selected by the **11b CHK_OPTN** field value, for verification control by the 16-byte forms of the special DIF CDBs.
- ?? The alternate set is selected for use, by the **10b** CHK_OPTN field control value, for verification control by the 16-byte forms of the special DIF CDBs.
- ?? The legacy set controls DIF verification during all normal/legacy read and verify operations (i.e., legacy CDBs).

The legacy write controls indicate the handling of the DIF in conjunction with a legacy write command.

The individual verification control flags have the following meanings:

The **REF_CK_x** flag controls the checking of the REF Tag sub-field against the nominal REF value. The nominal value is, according to the REF METHOD control and the specific CDB, one of the following two values:

- ?? The REF value indicated in a special DIF CDB (possibly incremented).
- ?? The lower 4 bytes of the LBA.

The **GRD_CK_x** flag indicates whether the guard sub-field of the DIF is checked against the calculated guard value.

The **ZAP_REF** flag indicates that the REF Tag value should be stored as zero (0) during a legacy write operation to indicate the source of the DIF.

The **MRK_GRD** flag indicates that the guard sub-field should be specially marked as part of a legacy write operation. When active, the guard is stored as zero (0000h) when using a checksum-based guard or the calculated CRC value XORed with BADBh for a CRC-based guard.

The **META TAG MASK x** fields provides masks used to limit the checking of the META Tag value. A value of 0000h effectively disables checking of the META Tag.

The **META TAG DEFAULT** field supplies the nominal (or generated) META Tag value used with legacy CDBs where there is no provision for a META Tag value.

The **STK_META**, **STK_REF**, and **STK_GRD** flags indicate whether the device has internal usage requirements for the META Tag, REF Tag, or Guard calculation method which would force it to "stack" DIFs (or provide some equivalent internal alternative) if the initiator alters those controls in byte 3 of the mode page. Such devices are generally intermediate controllers which make use of those fields in the DIF for internal purposes, e.g., it uses the META Tag as part of its block-state/status information (and/or as a logical LUN indicator), requires that the REF Tag be locked to LBA to allow it to internally verify block identity, or requires a specific Guard calculation method as part of certain internal algorithms (e.g., the use of checksum-based guards to close the RAID write-hole problem and/or to provide stripe protection). An intermediate controller should not require DIF-stacking for a **META ECHO** setting of **1** or a **REF METHOD** of **00b**.

The **DI_AVAIL** flag is informational only (marked as unchangeable). A device sets this flag if it internally provides some form of Data Integrity (DIF-based or some alternate method), e.g., a "hidden DIF" implementation.

Vendor specific fields are provided to allow for certain variances, e.g., little-endian semantics or explicit stacking of DIFs.

NOTE to reviewers: The following is to be extracted to a separate document:

The **LBA_40b** flag indicates whether the device will accept 40-bit LBAs. A device that does not provide this extension should force this flag to 0 (zero) and mark this bit as "unchangeable". If a device does provide for the 40-bit LBA extension, this flag should be marked as "changeable" and the setting of the flag indicates whether the extension is in force (flag is 1 [one]) or not (flag is 0 [zero]).

If the 40-bit LBA extension is not in force (i.e., **LBA_40b** is 0 [zero]), LBAs in CDBs are processed and checked normally, and the canonically reserved byte in 10, 12, and 16 byte CDBs remains "reserved".

If the **LBA_40b** flag is 1 (one), all CDBs with a 4-byte LBA are adjusted to accept a 5 byte (40 bit) LBA. The least significant 4 bytes of the LBA are provided by the existing 4 byte LBA field, while the most significant byte is provided in the canonically reserved byte in the CDB – This is byte is byte 6 in a 10-byte CDB[3], byte 10 in a 12-byte CDB, and byte 14 in a 16-byte CDB. This flag affects the following CDS:

> Lock Unlock Cache (10); Pre-Fetch (10); Read (10) and (12); Read Capacity (10); Read Long; Rebuild (16); Regenerate (16); Seek (10); Set Limits (10) and (12); Synchronize Cache (10); Verify (10) and (12); Write (10) and (12); Write and Verify (10) and (12); Write Long; Write Same (10); XDRead (10); XDWrite (10); XDWriteRead (10); and[4] XPWrite (10). In addition, the new DIF-aware CDBs with **CMD_FMT** equal to 0 (and thus 4 byte LBAs) are affected.

[3] For the 10-byte CDBs, there is some case for making byte 2 the most significant byte and byte 6 the least significant of a contiguous 5-byte LBA. However, the approach of making byte 6 the most significant would seem to have the smallest impact.

[4] XDWrite Extended (16) cannot be extended due to the use of byte 14 for the “Secondary Address”.

Legacy Write operations

When the device is formatted with attached DIFs, the DIF must be generated during a normal Write operation (i.e., existing write CDBs) under control of the Legacy Write control bits. The generation of the DIF is similar to the DIF generation described above during format operations:

?? The first (4 byte) sub-field of the DIF is either the lowest 4 bytes of the LBA or zero, depending on the **ZAP_REF** control flag.
?? The second (2 byte) sub-field is taken from the Data Integrity Mode page.
?? The guard sub-field is generated according to the guard selection controls in the Data Integrity Mode page. If the **MRK_GRD** flag is active, the guard sub-field is set to a special value to "mark" the block.

The device accepts the user data portion of each block, attaches the appropriate DIF, and writes the combined block to the media.

Legacy Write Same operations

During a Write Same operation on a device formatted with attached DIFs, a DIF must be generated and attached to each copy of the block, similar to a Write operation. If both the **PBDATA** and **LBDATA** flags are 0, the guard value can be calculated only once.

Legacy Read operations

When the device is formatted with attached DIFs, the DIF is checked and then stripped during normal read operations. The device reads the DIF-extended block from the media, verifies the DIF according to the Legacy set of flag settings in the Data Integrity Mode page, and sends the user data portion of the block (without the DIF) to the initiator.

[Notes to reviewers:

1. *Do we wish to "require" that a device keeps the DIF in it buffers and only checks and strips it as the last part of sending the block to the initiator?*
   Current position seems to be: It is nice to have, but not a requirement. Failure to do so weakens the integrity (especially with regard to the guard value).
2. *We need to specify the behavior when a DIF error is detected during a multi-block read. There seem to be two options: Terminate the read as soon as an error is recognized (send the block with the error but no more), OR, continue the read operation (sending all of the data blocks requested) and then indicate an error with the sense data indicating the first block in error and also indicating if there was only one block in error or if there were additional DIF errors. I tend to favor the latter approach, but I have not thought through all of the possible problems that might arise if there are other (non-DIF) errors and I could probably be talked into the former approach – This area needs to be discussed!!*
   Current position: The consensus seems to be to allow the device to terminate the operation, under control of the Read/Write Error Recovery mode page, when an error is encountered.

Legacy Verify operations

When the device is formatted with attached DIFs and a normal Verify operation is requested, the DIF is checked as described for a read operation and the user data part of the block is verified as specified by the existing Verify command requirements.

Legacy Write&Verify operations

The extension to the Write-and-Verify command is similar, but with the DIF generated as described for a Write operation and then verified as described above.

Write Long and Read Long (legacy)

These operations continue to access a complete block including any ECC (or other information on controllers that do not return ECC). However, if the device is formatted with attached DIFs, the size of the complete block includes the eight byte DIF (or 16 bytes for 2 DIFs in some cases) in addition to the ECC.

XOR Support operations

If the **STOR_DIF** flag is 1, the device is expected to reject the XOR support CDBs (REBUILD, REGENERATE, XDREAD, XDWRITE, XPWRITE, XDWRITEREAD, and XDWRITE EXTENDED).

DIF-aware operations

As a part of the Data Integrity Extensions, a number of new operations (CDBs) are defined to deal with blocks extended with DIFs. All of these operations should return ILLEGAL OPERATION/INVALID FORMAT sense data if they are issued to a device where the **STOR_DIF** flag is inactive. The additional operations extend the basic Read, Write, and Verify operations to deal directly with data blocks including an attached DIF, which is generally assumed to be accurate. In addition to transferring the DIF between the initiator and the device, the DIF-aware CDBs generally provide nominal values against which the REF Tag and META Tag sub-fields of the DIFs may be checked and controls to indicate if and when an error condition should be recognized as a result of an unexpected mis-match of the guard value or one of the tag values.

A device which implements the Data Integrity Extensions must provide support for either the 16 byte CDB formats and/or the 32 byte CDB formats (or both).

There are two additional CDBs, with common formats, provided for each operation type:

?? A 16-byte CDB. This form is provided for common use by the initiator but does not provide a full set of control options over the operation. There are two variant formats, one with a 4 byte LBA and an independent 4 byte nominal REF Tag value and a second format with an 8 byte LBA in which the Nominal REF Tag value is assumed to be the lower 4 bytes of the LBA. The format variant is selected by the **CMD_FMT** flag in byte 1, bit 7 of the CDB.

?? A 32-byte CDB that provides full control over the operation. In this format, many of the Mode page controls are duplicated and the values provided in the CDB take precedence. If the device does not support the control combination specified in the CDB, it is expected to reject the command.

The 16-byte CDBs use the following templates[5]. When the **CMD_FMT** flag is 0 (zero):

Bit / **Byte**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	OPERATION CODE (TBDh)							
1	CMD_ FMT (0)	CHK_OPTN		[---Command specific flags---] {See the individual command descriptions}				
2	(MSB)							
3	LOGICAL BLOCK ADDRESS (LBA)							
4								
5								(LSB)
6	(MSB)							
7	(Nominal) REF TAG							
8								
9								(LSB)
10	(MSB)	(Nominal) META TAG						
11								(LSB)
12	(MSB)	TRANSFER LENGTH						
13								(LSB)
14	RESERVED							
15	CONTROL							

[5] This approach requires the use of 5 of the 16-byte Operation codes. This can be reduced to only 2 codes by combining the DIF-Write, DIF-Verify, DIF-Write&Verify, and DIF-Write Same CDBs into a single Operation Code with a sub-operation field, as follows:

1. Move the **CHK_OPTN** field to the (two) reserved bits in the "Command Specific Flag" area. The placement (unfortunately) varies according to the specific sub-operation., using bits 1 and 2 for the DIF-Write sub-operation, bits 2 and 3 for the DIF-Verify and DIF-Write&Verify sub-operations, and bits 3 and 4 for the DIF-Write Same sub-operation.
2. Use bits 5 and 6 to indicate the sub-operation, 00b for DIF-Write, 01b for the DIF-Write Same, and 10b and 11b for the DIF-Verify and DIF-Write&Verify operations.
3. For uniformity, the format of the DIF-Read CDB should be altered to conform to the format of the DIF-Write CDB, displacing **CHK_OPTN** to bits 1 and 2 and making bits 5 and 6 "reserved".

Format when the CMD_FMT flag is 1 (one):

Bit / Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	OPERATION CODE (TBDh)							
1	CMD_ FMT (1)	CHK_OPTN		[---Command specific flags---] {See the individual command descriptions}				
2	(MSB)							
3	LOGICAL BLOCK ADDRESS (LBA)							
4	[The nominal REF TAG is supplied by the lower 4 bytes]							
5								
6								
7								
8								
9								(LSB)
10	(MSB)							
11	(Nominal) META TAG							(LSB)
12	(MSB)							
13	TRANSFER LENGTH							(LSB)
14	RESERVED							
15	CONTROL							

The **CMD_FMT** flag selects between the two 16-byte formats of the special DIF CDBs. If the **CMD_FMT** flag is 0 (zero), the 16-byte format contains a 4 byte LBA and a separate 4 byte REF tag checking value. If the **CMD_FMT** flag is 1 (one) the 16-byte format contains an 8 byte LBA and the REF tag checking value is assumed to be equal to the lower 4 bytes of the LBA.

The **CHK_OPTN** field indicates the checking to be performed against the attached DIFs during the Read operation. The values of this field are as follows:

CHK_OPTN	**DIF Checking performed**
00	None
01	Guard value only
10	As indicated by the Alternate Verification flags in the Mode page
11	As indicated by the Primary Verification flags in the Mode page

The “Command Specific Flags” vary according to the specific operation and are identical to the flag definitions of bits 0 through 4 in the corresponding non-DIF-aware command. The specific flag bit definitions are indicated in the descriptions (below) of the individual DIF-aware operations.

The 32-byte format of the DIF-aware CDBs use the following template:

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (7Fh)							
1	CONTROL							
2	Reserved							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved							
7	ADDITIONAL CDB LENGTH (18h)							
8	(MSB) SERVICE ACTION (TBDh)							
9								(LSB)
10	Reserved			[---Command specific flags---] {See the individual command descriptions}				
11	Vendor Specific		REF CHECK	GUARD CHECK	Reserved			
12	(MSB) LOGICAL BLOCK ADDRESS							
13								
14								
15								
16								
17								
18								
19								(LSB)
20	(MSB) (Nominal) REF TAG							
21								
22								
23								(LSB)
24	(MSB) (Nominal) META TAG							
25								(LSB)
26	(MSB) META TAG MASK							
27								(LSB)
28	(MSB) TRANSFER LENGTH							
29								
30								
31								(LSB)

The **REF CHECK**, and **GUARD CHECK** flags correspond to the verification control flags in the Mode page and specify the verification checks to be performed for this operation. The **META TAG MASK** is always applied, but since a value of 0000h will effectively obviate the mask functionality, so there is no need to have a separate Mask control flag.

[Note to Reviewers: The 4 reserved bits in byte 11 could possibly be used to specify the guard calculation or exclusion count, but this does not seem to be a particularly useful feature to provide – Generally in the cases where it might be useful, it is probably simpler to suggest that the GUARD CHECK flag be set to 0 and any special verification be done in code instead.]

DIF-Read operations

The new DIF-Read CDB reads the complete data block (including the DIF) over the interface. The ”Command Specific Flags” for this operation are as follows:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1:	---			DPO	FUA	Reserved		RELADR

During a DIF-Read, the device should send the data blocks unmodified, except as allowed and indicated by the Mode page controls. However, the device is expected to verify the DIF according to the information and flags in the CDB and in the Data Integrity Mode page (if selected, i.e., for **CHK_OPTN** values **10b** and **11b**) and indicate an error at the end of the transfer if a DIF error is recognized.

DIF-Write operations

The new DIF-Write CDBs cause the complete data block, including the 8 byte DIF, to be accepted by the device and written to the media. The ”Command Specific Flags” for this operation are as follows:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1:	---			DPO	FUA	Reserved		RELADR

The device should not modify the DIF in any way before writing it to the media, but is expected to check the DIFs as they are accepted from the initiator and indicate any DIF-errors it recognizes in the transmitted data. As for a DIF-Read, the checking is controlled by information and flags in the CDB and in the Data Integrity Mode page. If a DIF-error is recognized, the device is expected to complete the operation and indicate the error at the end with sense data indicating the first block in error.

[Notes to reviewers:

1. *Again, there is the option of stopping on the first block in error rather than completing the operation (should we have an error control flag in the Data Integrity Mode page??).*
   The current consensus seems to be that the device may terminate the operation whenever it detects a DIF error. In particular, the action on encountering and error should be controlled by the Read/Write Error Recovery mode page.
2. *A good case can be made for allowing the device to generate and fill-in some of the DIF sub-fields since this would help to reduce the firmware/software overheads. I think we have to have the full transmission possibility for flexibility and completeness. However, a good case can be made for allowing the controller in the initiator to fill in certain sub-fields of the DIF, and it may be worthwhile to allow the device to fill in such fields as well in certain cases.]*
   This might be a possible usage for the reserved bits in byte 11 of the 32-byte CDB format.

DIF-Verify operations

The new DIF-Verify operation is used to verify that the DIF areas are as expected as well as (possibly) verifying the user data. The “Command Specific Flags” for this operation are as follows:

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1:	---			DPO	Reserved		BYTCHK	RELADR

During a DIF-Verify, the device accepts the complete data block, including the DIF, and checks it against the appropriate data block from the media.

DIF-Write&Verify operations

The CDBs for the DIF-Write&Verify operation are similar to the DIF-Verify CDBs and the “Command Specific Flags” have the same layout.

DIF-Write Same operations

The DIF-Write Same operation accepts a single block of data, with a DIF, from the initiator and duplicates that block through the indicated range of blocks. The “Command Specific Flags” for this operation are as follows:

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1:	---			Reserved		PBDATA	LBDATA	RELADR

In addition to the possible modifications of the data indicated by the PBDATA and LBDATA flags, the device may need to make the following alterations to the DIF on a block-by-block basis:

?? The REF Tag value may need to be incremented on a per-block basis.
?? The guard value may need to be adjusted due to any changes in the data block that result from the PBDATA or LBDATA modifications to the block.

[Note to reviewers: There was some discussion as to whether we should disallow the PBDATA and LBDATA flags in the DIF-Write Same, both to avoid the extra overhead of modifying the guard value and because the REF Tag serves essentially the same purpose. However, the code to update the guard value is still required to handle the non-DIF Write Same CDB (if the flags are allowed), and an implementation can always reject the usage of those flags, so it seems reasonable to provide them here.]

Algorithm for calculating the Checksum-based Guard Value:

The following C code describes the calculation of the checksum-based Guard value. Note that “len” is expected/required to be even (actually, it will normally be a multiple of 4):

```
unsigned short ChkSum( register unsigned char * data,
                       register int             len)
{
    register unsigned long sum = 0xFFFF;

    /* Infinite loop with mid-point “break” exit: */
    do
    {
        sum <<= 1;
        sum += *((unsigned short*)data);
        data += 2;    /*processing 2 bytes at a time */
        if( (len -= 2) & 0x0F)   /* loop if not *16 */
            loop;

        /* Wrap carries down for 1’s complement add*/
        sum = (sum & 0xFFFF) + (sum >> 16);

        if( len == 0)   /* done? */
            break;
    }
    return (unsigned short)
           ((sum & 0xFFFF) + (sum >> 16));
}  /* END of checksum calculation routine */
```

This routine produces the same (in memory) result under both “little endian” and “big endian” processor architectures when the result is stored in memory, i.e., the first result byte in memory will have the same value in either architecture and the second byte in memory will have the same value[6]. In particular, on a “big endian” architecture, the nominal data layout (for a 512 byte plus DIF data block in 2-byte words) is the following:

msb	lsb	msb	lsb	...	msb	lsb	msb			lsb	msb	lsb	msb	lsb
Word 0		Word 1		...	Word 255		REF Tag				META Tag		Guard	
													AA	BB

While, on a “little endian” architecture, the nominal data layout is the following:

lsb	msb	lsb	msb	...	lsb	msb	lsb			msb	lsb	msb	lsb	msb
Word 0		Word 1		...	Word 255		REF Tag				META Tag		Guard	
													AA	BB

Because the inputs to the calculation are “swapped” in the “little endian” architecture, the calculated value is also “swapped” (multiplied by 256 modulo 2^{16}-1), and the resulting value is “re-swapped” when stored in memory, resulting in the same values in byte order.

[6] This is a result of the special properties of 16-bit 1’s complement (i.e., modulo 2^{16}-1) arithmetic. The (relative) byte swap between “little endian” and “big endian” architectures is equivalent to a multiplication by 256 modulo 2^{16}-1, and 256*256 = 1 modulo 2^{16}-1.

Examples of the Checksum-based Guard

These examples all use a data block containing the following data bytes (in hexadecimal):

01, 35, 2C, 57, 68, 4B, 59, 97, 74, 82, followed by 492 bytes of 00.

When calculated on a “big endian” architecture, the calculations are as follows:

Operation	**Checksum**
seed	FFFF
rotate	FFFF
data: 01,35	0135
rotate	026A
data: 2C,57	2EC1
rotate	5D82
data: 68,4B	C5CD
rotate	4B5B
data: 59,97	A4F2
rotate	49E5
data: 74,82	BE67
rotate * 11	3DF3
rotate * 480	3DF3
FINAL:	3DF3
Stored in Memory	3D,F3

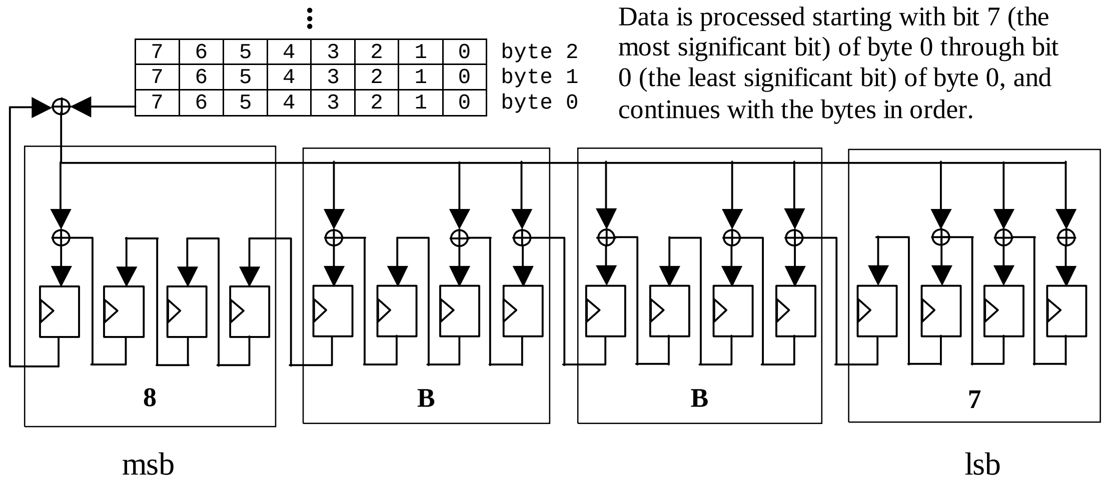
When calculated on a “little endian” architecture, the calculations are as follows:

Operation	**Checksum**
seed	FFFF
rotate	FFFF
data: 35,01	3501
rotate	6A02
data: 57,2C	C12E
rotate	825D
data: 4B.68	CDC5
rotate	5B4B
data: 97,59	F2A4
rotate	E549
data: 82,74	67BE
rotate * 11	F33D
rotate * 480	F33D
FINAL:	F33D
Stored in Memory	3D,F3

As can be seen from these examples, the value stored in memory (and attached as the guard value to the data block) is the same in both architectural models,

Algorithm for calculating the CRC-based Guard value:

The CRC-based Guard value is nominally calculated a bit–at-a-time by the following logic circuit. The initial feedback register value is 0000h:



Data is processed starting with bit 7 (the most significant bit) of byte 0 through bit 0 (the least significant bit) of byte 0, and continues with the bytes in order.

An implementation may provide an equivalent computation that produces the same value. An alternate implementation may well process the data a byte at a time, 2 bytes at a time, or in even larger blocks of data per (clock) step.

As an example for verification purposes, the CRC computed, starting with a seed of FFFFh, for the following 512 byte data block:

index	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
00x	FF	FE	FD	FC	FB	FA	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
01x	EF	EE	ED	EC	EB	EA	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0
02x	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
03x	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
...								...								
1Fx	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

is the value 69CDh and the DIF will be as follows:

(MSB) REF Tag (LSB)	(m) META Tag (l)	69h	CDh