Trim Command – General Benefits for Hard Disk Drives

Author | Wes McMillen, Western Digital Corporation
Introduction

The purpose of a trim command is to allow the host to provide LBA/data usage information that allows the storage device to become more performant. The trim command provides input from the host to let the HDD know what sectors no longer contain user data. Thus, trimming data allows the HDD to minimize background data movement and other disk background activities that can impact host performance. Executing the trim command during opportune scenarios allows for additional host features such as additional cameras, auxiliary streams, picture streams, and playback streams.

Requested Trim Scenarios

HDD Initialization

Western Digital requests the host execute a TRIM across the entire range of the drive using the DATA SET MANAGEMENT XL’s TRIM operation when the system is going through an HDD initialization operation.

This step is critical if the user reuses a drive that was previously in an overstressed environment where its performance may have been degraded. The TRIM operation will reset and re-initialize the drive to a clean state.

Opening of an Allocation Unit (AU)

When the system opens a new allocation unit to write IPC data, Western Digital requests that the LBA range for the allocation unit be trimmed. If a sub-stream/auxiliary stream requires opening a new allocation unit, a TRIM of the range for the auxiliary/sub-stream is also required.

Executing the TRIM over allocation units before they are used allows the device to clean up any old data that was left over when the allocation unit is reused. This enables a significant reduction in background data movement which frees up more disk bandwidth for more features such as additional cameras, auxiliary streams, picture streams, and playback streams. Even when the device has been previously completely trimmed, any write to an allocation unit must first be preceded by a trim for the AU.

Summary of Trim XL and AU suggestions:

- All stream data writes in a host AU must be preceded by a Trim (XL) command.
- Host OS meta data areas do not need a trim.
- Trim (XL) the entire drive before performing high level system format.
- Trim LBA and range must be 4K aligned.
- Only use on range for the trim of the AU (not multiple ranges that combined equal the AU).

Additional host OS suggestions:

- Use a large Allocation Unit (AU).
  Recommend at least 1GiB/AU. 2GiB would be preferable.
- Always allocate a full AU per stream.
  Do not partially allocate AUs at the end of the HDD LBA space.
- AUs must represent the same LBA space after every wrap.
- Co-locate all video meta-data in the same video stream AU LBA space.
- Co-locate all system level meta-data in a confined area in the lower range of the LBA space.
- Locate video meta-data as a sequential stream (either forward or reverse).
- Do not backup and rewrite LBAs in an AU.
  Write the video streams purely sequential. It is understood that the video meta-data
streams will typically rewrite data and this practice is acceptable.

- Use a large and consistent block count sizes for the video stream write commands. This allows the HDD to identify this as a video record write command.
- Use significantly smaller block count sizes for meta-data write commands.
  This allows the HDD to identify this write data as meta-data.
- Write all streams (video/aux/picture) forward sequentially. The exception for this rule is a reverse video meta-data stream.
- Take care on placement of buzzers in the SVR box used for alerts. Improper placement or buzzer frequency could impact drive performance. In general, eliminate vibration as much as possible.
- Limit use of ATA Flush commands to the bare minimum.
- Incorporate the use of the ATA NOP command to inform the HDD of a system buffer under-run/over-run condition. This helps greatly with system validation and debugging.
- Incorporate the use of ATA security (password) for unlocking and system identification for the HDD.
- All data must be 4K aligned for optimum performance.
- NCQ 6G SATA support.
- Trim LBA and range must be 4K aligned.

TRIM DSM (Legacy) vs TRIM DSM XL

Trim is supported by either legacy TRIM Dataset Management (Opcode 06h) or TRIM Dataset Management XL (Opcode 07h). The main problem with the legacy TRIM DSM is that the TRIM RANGE LENGTH is limited to 16-bit vs 64-bit for TRIM DSM XL. Trimming large amount of data potentially requires hundreds if not thousand instances of legacy TRIM DSL commands to be issued vs one or a few for TRIM DSM XL. HDD processing of thousand instances of either trim ranges or legacy TRIM DSM can add tens of seconds to the trimming process. For this reason, WDC requests the host to implement TRIM with DSM XL.

RAID Rebuild

A RAID rebuild is the data reconstruction process that occurs when a hard disk drive needs to be replaced. A Trim command must be issued to restore original performance.
Implementation

The Trim functionality should be implemented in Linux Application Layer. In the Linux application boot-up, it should check if Data Set Management XL (DSM XL) trim is supported. The support flag for this drive should be stored in a global variable in the Application Layer.

Data Set Management XL (DSM XL) support bit is specified in ACS-4 documentation. To check if DSM XL is supported, query Log Address 30H (Identify Device Data Log), Page 03H (Supported Capabilities), Qword (byte offset 8-15), Bit 50.

Before the application sends a DSM XL Trim command to a drive, it should check the global variable to see if the drive supports trim.

At the end of this document is sample code for trim. To check if the DSM XL Trim command is supported, please refer to the function `is_dsm_xl_supported()`.

Important: It is strongly recommended to check the feature support once when the application starts, rather than checking the feature support every time a trim is issued. This will reduce unnecessary overhead.

On a HDD hot-plug condition, the DSM XL support bit must be rechecked and the global variable re-initialized.

DSM XL is a non-queued command. If the OS is operating in an NCQ environment, all of the queued commands must complete before this non-queued command is issued. Otherwise, there will be a queue command intermix error.

Error Handling

Check for DSM XL support before issuing the command to prevent the OS from executing exception handling. This can occur due to an error condition generated for drives that do not support DSM XL Trim.

On a DSM XL Trim command error, the application layer may choose to dump failure information, such as if DSM XL Trim is supported, Trim Start LBA and Trim length.

Identify Device

7.1 Log Address 30H (Identify Device Data)

<table>
<thead>
<tr>
<th>Log Address</th>
<th>Log Name</th>
<th>Feature Set</th>
<th>Support</th>
<th>R/W</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Ah.. 1Fh</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20h</td>
<td>Obsolette</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21h</td>
<td>Write Stream Error Log</td>
<td>Streaming</td>
<td>F</td>
<td>RO</td>
<td>GPL b</td>
</tr>
<tr>
<td>22h</td>
<td>Read Stream Error Log</td>
<td>Streaming</td>
<td>F</td>
<td>RO</td>
<td>GPL b</td>
</tr>
<tr>
<td>23h</td>
<td>Obsolete</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24h</td>
<td>Current Device Internal Status Data log</td>
<td>None</td>
<td>O</td>
<td>RO</td>
<td>GPL b</td>
</tr>
<tr>
<td>25h</td>
<td>Saved Device Internal Status Data log</td>
<td>None</td>
<td>O</td>
<td>RO</td>
<td>GPL b</td>
</tr>
<tr>
<td>26h.. 2Eh</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2Fh</td>
<td>Set Sector Configuration</td>
<td>None</td>
<td>RO</td>
<td>GPL b</td>
<td></td>
</tr>
<tr>
<td>30h</td>
<td>IDENTIFY DEVICE data</td>
<td>None</td>
<td>M</td>
<td>RO</td>
<td>GPL, SL</td>
</tr>
<tr>
<td>31h.. 7Fh</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80h.. 9Fh</td>
<td>Host Specific</td>
<td>SMART</td>
<td>M</td>
<td>R/W</td>
<td>GPL, SL</td>
</tr>
<tr>
<td>A0h.. D7h</td>
<td>Device Vendor Specific</td>
<td>SMART</td>
<td>O</td>
<td>VS</td>
<td>GPL, SL</td>
</tr>
<tr>
<td>E0h</td>
<td>SCT Command/Status</td>
<td>SCT</td>
<td>F</td>
<td>R/W</td>
<td>GPL, SL</td>
</tr>
<tr>
<td>E1h</td>
<td>SCT Data Transfer</td>
<td>SCT</td>
<td>F</td>
<td>R/W</td>
<td>GPL, SL</td>
</tr>
<tr>
<td>E2h.. FFh</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.2 Log Address 30H (Identify Device Data)

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>List of supported pages (see 9.10.2)</td>
<td>M</td>
</tr>
<tr>
<td>01h</td>
<td>Copy of IDENTIFY DEVICE data (see 7.13.6)</td>
<td>M</td>
</tr>
<tr>
<td>02h</td>
<td>Capacity (see 9.10.4)</td>
<td>M</td>
</tr>
<tr>
<td>03h</td>
<td>Supported Capabilities (see 9.10.5)</td>
<td>M</td>
</tr>
<tr>
<td>04h</td>
<td>Current Settings (see 9.10.6)</td>
<td>M</td>
</tr>
<tr>
<td>05h</td>
<td>ATA Strings (see 9.10.7)</td>
<td>M</td>
</tr>
<tr>
<td>06h</td>
<td>Security (see 9.10.8)</td>
<td>M</td>
</tr>
<tr>
<td>07h</td>
<td>Parallel ATA (see 9.10.9)</td>
<td>P</td>
</tr>
<tr>
<td>08h</td>
<td>Serial ATA (see 9.10.9)</td>
<td>S</td>
</tr>
<tr>
<td>09h</td>
<td>Reserved for ZAC</td>
<td>See ZAC</td>
</tr>
<tr>
<td>0Ah.. FFh</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
7.3 Supported Capabilities Page (DSM XL Support Bit)

<table>
<thead>
<tr>
<th>Offset Type Content</th>
<th>Bit Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8..15 QWord</td>
<td>Supported Capabilities</td>
</tr>
<tr>
<td></td>
<td>Bit Description</td>
</tr>
<tr>
<td>63</td>
<td>Shall be set to one</td>
</tr>
<tr>
<td>62:55</td>
<td>Reserved</td>
</tr>
<tr>
<td>54</td>
<td>ADVANCED BACKGROUND OPERATION SUPPORTED bit</td>
</tr>
<tr>
<td>53</td>
<td>PERSISTENT SENSE DATA REPORTING bit</td>
</tr>
<tr>
<td>52</td>
<td>SFF-8447 REPORTING bit</td>
</tr>
<tr>
<td>51</td>
<td>DEFINITIVE ENDING PATTERN SUPPORTED bit</td>
</tr>
<tr>
<td>50</td>
<td>DATA SET MANAGEMENT XL SUPPORTED bit</td>
</tr>
<tr>
<td>49</td>
<td>SET SECTOR CONFIGURATION SUPPORTED bit</td>
</tr>
<tr>
<td>48</td>
<td>ZERO EXT SUPPORTED bit</td>
</tr>
<tr>
<td>47</td>
<td>SUCCESSFUL NCQ_COMMAND SENSE DATA SUPPORTED bit</td>
</tr>
<tr>
<td>46</td>
<td>DLC SUPPORTED bit</td>
</tr>
<tr>
<td>45</td>
<td>REQUEST SENSE DEVICE FAULT SUPPORTED bit</td>
</tr>
</tbody>
</table>

Data Set Management XL
The Data Set Management XL command (ATA Opcode 07h) provides information (e.g., file system information) that the device may use to optimize its operations.

Inputs
See the following table for a Data Set Management XL Trim Command:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE</td>
<td>Bit Description</td>
</tr>
<tr>
<td>15:8</td>
<td>DSM Function Field</td>
</tr>
<tr>
<td>7:1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>TRIM bit</td>
</tr>
<tr>
<td>COUNT</td>
<td>Number of 512-byte blocks to be transferred. The value zero is reserved.</td>
</tr>
<tr>
<td>LBA</td>
<td>if the TRIM bit is set to one, reserved. If the TRIM bit is cleared to zero, defined by the DSM FUNCTION field.</td>
</tr>
<tr>
<td>DEVICE</td>
<td>Bit Description</td>
</tr>
<tr>
<td>7</td>
<td>Obsolete</td>
</tr>
<tr>
<td>6</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>Obsolete</td>
</tr>
<tr>
<td>4</td>
<td>Transport Dependent</td>
</tr>
<tr>
<td>3:0</td>
<td>Reserved</td>
</tr>
<tr>
<td>COMMAND</td>
<td>7:0 07h</td>
</tr>
</tbody>
</table>

Host Inputs to Device Data Structure
Data Set Management XL Request Data is a list of one or more XL LBA Range Entry pages (see table below).

If the TRIM bit is set to one, individual XL LBA Range Entries may specify LBA ranges that overlap and are not required to be sorted.

XL LBA Range Entry

<table>
<thead>
<tr>
<th>Offset Type Description</th>
<th>Bit Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..15 DQWord Entry 0</td>
<td>127:64 RANGE LENGTH field</td>
</tr>
<tr>
<td></td>
<td>63:48 Reserved</td>
</tr>
<tr>
<td></td>
<td>74:0 LBA VALUE field</td>
</tr>
<tr>
<td>16..31 DQWord Entry 1</td>
<td>127:64 RANGE LENGTH field</td>
</tr>
<tr>
<td></td>
<td>63:48 Reserved</td>
</tr>
<tr>
<td></td>
<td>47:0 LBA VALUE field</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>496..511 DQWord Entry 31</td>
<td>127:64 RANGE LENGTH field</td>
</tr>
<tr>
<td></td>
<td>63:48 Reserved</td>
</tr>
<tr>
<td></td>
<td>47:0 LBA VALUE field</td>
</tr>
</tbody>
</table>

RANGE LENGTH Field
The RANGE LENGTH field specifies the number of logical sectors in the LBA range. If the RANGE LENGTH field is set to 0h, the entry shall be ignored.

LBA VALUE Field
The LBA Value field specifies the starting LBA of the LBA range. If the LBA VALUE plus the range is greater than the accessible capacity, the device shall return command aborted.
Sample Source Code

/* Sample code how to check if DSM XL is supported. It issues a DSM XL to trim the entire HDD if DSM XL is supported.

The function is_dsm_xl_supported() returns 1 (if DSM XL is supported) and 0 (if DSM XL is not supported). This state is then saved. Next time a trim issued, the application will check against the saved state versus issuing the Read Log command again.

The check for DSM XL supported should be part of the Application Bootup process.
*/

Sample Source Code:

```c
#include <ctype.h>
#include <fcntl.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <scsi/sg.h>
#include <unistd.h>

// See ACS-4 DATA SET MANAGEMENT XL command
#define DSM_TRIM_XL_ENTRY_SIZE (16)
#define ATA_SECTOR_SIZE (512)
#define SGIO_TIMEOUT (1000 * 60)
#define MAX_DEVICES (1024)

// global variables to store info per device
struct device_data_t {
    char path[FILENAME_MAX];
    uint64_t total_lbas;
    uint8_t supported;
};

static struct device_data_t *devices[MAX_DEVICES];
static uint32_t_device_count = 0;

// Swap bytes from NULL terminated string. Useful for ATA IDENTIFY DEVICE strings
char *byteswap(char *str) {
    size_t i;
    if (!str) {
        return 0;
    }
    for (i = 0; i <= strlen(str) && i + 1 <= strlen(str); i += 2) {
        char a, b;
        a = str[i];
        b = str[i + 1];
        str[i] = b;
        str[i + 1] = a;
    }
    return str;
}

// trim leading/trailing spaces from string
char *trim(char *str) {
    size_t len;
    int i;
    if (!str) {
        return 0;
    }
    for (i = 0; i <= (int)strlen(str) && str[i] == ' '; ++i);
    len = strlen(str + i);
    memmove(str, str + i, len);
    str[len] = 0;
    for (i = strlen(str); i >= 0; i--) {
        char *c = str + i;
        if (isspace(*c) || *c == '\0') {
            *c = 0;
        }
    }
    return str;
}
```

/* Sample code how to check if DSM XL is supported. It issues a DSM XL to trim the entire HDD if DSM XL is supported.

The function is_dsm_xl_supported() returns 1 (if DSM XL is supported) and 0 (if DSM XL is not supported). This state is then saved. Next time a trim issued, the application will check against the saved state versus issuing the Read Log command again.

The check for DSM XL supported should be part of the Application Bootup process.
*/
void le_u64_to_array(uint64_t value, uint8_t* buffer) {
    int i;
    for (i = 0; i < 8; i++) {
        *buffer++ = value & 0xFF;
        value >>= 8;
    }
}

uint16_t array_to_le_u16(uint8_t word, const uint8_t* const buffer, size_t size) {
    if (!buffer || size == 0 || word * 2 + 1 >= size) {
        fprintf(stderr, "invalid parameter(s)\n");
        return 0;
    }
    return (uint16_t)buffer[word * 2] | ((uint16_t)buffer[word * 2 + 1] << 8);
}

uint32_t array_to_le_u32(uint8_t dword, const uint8_t* const buffer, size_t size) {
    if (!buffer || size == 0 || dword * 4 + 3 >= size) {
        fprintf(stderr, "invalid parameter(s)\n");
        return 0;
    }
    return (uint32_t)buffer[dword * 4] | ((uint32_t)buffer[dword * 4 + 1] << 8) | ((uint32_t)buffer[dword * 4 + 2] << 16) | ((uint32_t)buffer[dword * 4 + 3] << 24);
}

uint64_t array_to_le_u64(uint8_t qword, const uint8_t* const buffer, size_t size) {
    if (!buffer || size == 0 || qword * 8 + 7 >= size) {
        fprintf(stderr, "invalid parameter(s)\n");
        return 0;
    }
    return (uint64_t)buffer[qword * 8] | ((uint64_t)buffer[qword * 8 + 1] << 8) | ((uint64_t)buffer[qword * 8 + 2] << 16) | ((uint64_t)buffer[qword * 8 + 3] << 24) | ((uint64_t)buffer[qword * 8 + 4] << 32) | ((uint64_t)buffer[qword * 8 + 5] << 40) | ((uint64_t)buffer[qword * 8 + 6] << 48) | ((uint64_t)buffer[qword * 8 + 7] << 56);
}

int scsi_command(int fd, int direction, uint8_t* cdb, size_t cdb_sz, uint8_t* buffer, size_t buffer_sz, uint8_t* sense, size_t sense_sz) {
    int ret;
    int i;
    return ret;
}
sg_io_hdr_t sgio;

// clear sense data
memset(sense, 0, sense_sz);

// Setup SG_IO pass-through
memset(&sgio, 0, sizeof(sgio));
sgio.interface_id = 'S';
sgio.dxfer_direction = direction;
sgio.dxfer_len = buffer_sz;
sgio.dxferp = buffer;
sgio.cmd_len = cdb_sz;
sgio.cmdp = cdb;
sgio.mbx_sb_len = sense_sz;
sgio.sbp = sense;
sgio.timeout = SGIO_TIMEOUT;
ret = ioctl(fd, SG_IO, &sgio);
if (ret) {
    fprintf(stderr, "ioctl failed (ret = %d)\n", ret);
} else if (sgio.status & 0x01 || sgio.driver_status) {
    fprintf(stderr, "device error (status = 0x%x; driver_status = 0x%x)\n",
            sgio.status, sgio.driver_status);
    for (i = 0; i < sgio.sb_len_wr; i++) {
        fprintf(stderr, " %02x", sense[i]);
    }
    fprintf(stderr, "\n");
    ret = 1;
}
return ret;

/**
 * Use SG_IO to send READ LOG EXT command to ATA device.
 * parameters:
 *  fd            Send command to this file description
 *  log_address   ATA spec: log address
 *  page_number   ATA spec: page number
 *  buffer        Returns id buffer device data
 *  buffer_sz     Size of buffer (should be a multiple of 512)
 * returns:
 *  error        0 upon success
 */
int ata_read_log_ext(int fd, uint8_t log_address,
                     uint16_t page_number,
                     uint8_t* buffer, size_t buffer_sz) {
    uint8_t_t cdb[16];
    uint8_t_t sense[32];
    uint16_t features = 0;
    uint16_t log_page_count = buffer_sz / 512;
    uint64_t lba = (((uint64_t)page_number & 0xFF00) << 24) |
                          ((page_number & 0x00FF) << 8) |
                          (log_address);
    if (fd < 0 || !buffer || buffer_sz < ATA_SECTOR_SIZE) {
        fprintf(stderr, "invalid parameter(s)\n");
        return -1;
    }
    // Set CDB - SAT spec, section 12.2.2.3
    cdb[0] = 0x85;  // ATA pass-through (16)
    cdb[1] = 0x09;  // PROTOCOL (bit 4:1 = PIO Data-In [4]); EXT command (bit 0 = 1)
    cdb[2] = 0x0E;  // OFFLINE (bit 7:6 = 0); CK_COND
    (bit 5 = 0); T_TYPE (bit 4 = 0); T_DIR (bit 3 = 1);
    BYTE_BLOCK (bit 2 = 1); T_LENGTH (bit 1:0 = 2)
    cdb[3] = (features >> 8) & 0xFF;
    cdb[4] = features & 0xFF;
    cdb[5] = (log_page_count >> 8) & 0xFF;
    cdb[6] = log_page_count & 0xFF;
    cdb[7] = (lba >> 24) & 0xFF;  // LBA (31:24)
    cdb[8] = lba & 0xFF;         // LBA (7:0)
    cdb[9] = (lba >> 32) & 0xFF;
    cdb[10] = (lba >> 8) & 0xFF;
    cdb[11] = (lba >> 40) & 0xFF;
    cdb[12] = (lba >> 16) & 0xFF;
    cdb[13] = 0x00;              // DEVICE
    cdb[14] = 0x2f;              // COMMAND = READ LOG EXT
    cdb[15] = 0x00;
    return scsi_command(fd, SG_DXFER_FROM_DEV, cdb, sizeof(cdb), buffer, buffer_sz, sense, sizeof(sense));
}

/**
 * Use SG_IO to send IDENTIFY DEVICE command to ATA device.
 * parameters:
 *  fd            Send command to this file description
 *  buffer        Returns id buffer device data
 *  buffer_sz     Size of buffer (must be at least 512 bytes)
 * returns:
 *  error        0 upon success
 */
int ata_identify_device(int fd, uint8_t_t* buffer, size_t buffer_sz) {
if (fd < 0 || !buffer || buffer_sz < ATA_SECTOR_SIZE) {
    fprintf(stderr, "invalid parameter(s)\n");
    return -1;
 }

// Set CDB - SAT spec, section 12.2.2.2
cdb[0] = 0xA1;  // ATA pass-through (12)
cdb[1] = 0x08;  // PROTOCOL (bits 4:1 = PIO Data-In[4])
cdb[2] = 0x0E;  // T_DIR (bit 3 = 1); BYTE_BLOCK
(bit 2 = 1); T_LENGTH (bits 1:0 = 2)
cdb[3] = 0x00; // FEATURES (7:0)
cdb[4] = 0x01; // COUNT (7:0)
cdb[5] = 0x00; // LBA (7:0)
cdb[6] = 0x00; // LBA (15:8)
cdb[7] = 0x00; // LBA (23:16)
cdb[8] = 0x00; // DEVICE

cdb[9] = 0xEC; // COMMAND = IDENTIFY DEVICE
cdb[10] = 0x00; // reserved
cdb[11] = 0x00; // CONTROL

return scsi_command(fd, SG_DXFER_FROM_DEV, cdb, sizeof(cdb), buffer, buffer_sz, sense, sizeof(sense));
}

/**
* Use SG_IO to send ATA data set management trim XL
* command to ATA device.
* parameters:
*   fd          Send command to this file desciption
*   entry       Pointer to a dsm_trim_xl_range_entry array
*   entry_size  The number of valid entries
* * returns:
*   * error      0 upon success
*
*/
int ata_trim_xl(int fd, dsm_trim_xl_range_entry *entry, int entry_size) {
    int ret;
    int i;
    uint8_t cdb[16];
    uint8_t sense[32];
    uint8_t *buffer;
    size_t buffer_sz;
    uint64_t lba = 0;
    uint16_t count;

    if (fd < 0 || !entry || !entry_size) {
        fprintf(stderr, "invalid parameter(s)\n");
        return -1;
    }

    // Prepare buffer for DSM TRIM XL
    count = (entry_size * DSM_TRIM_XL_ENTRY_SIZE) + ATA_SECTOR_SIZE - 1) / ATA_SECTOR_SIZE;
    buffer_sz = count * ATA_SECTOR_SIZE;
    buffer = malloc(buffer_sz);
    if (!buffer) {
        fprintf(stderr, "out of memory\n");
        return -1;
    }
    memset(buffer, 0, buffer_sz);

    for (i = 0; i < entry_size; i++) {
        le_u64_to_array(entry[i].lba_value, buffer + i * DSM_TRIM_XL_ENTRY_SIZE);
        le_u64_to_array(entry[i].range_length, buffer + sizeof(uint64_t) + i * DSM_TRIM_XL_ENTRY_SIZE);
    }

    // Set CDB - SAT spec, section 12.2.2.3
    cdb[0] = 0x85;  // ATA pass-through (16)
cdb[1] = 0x0D;  // PROTOCOL (bit 4:1 = DMA[6]); EXT
    command (bit 0 = 1)
cdb[2] = 0x02;  // OFFLINE (bit 7:6 = 0); CK_COND
(bit 5 = 0); T_TYPE (bit 4 = 0); T_DIR (bit 3 = 0);
cdb[3] = 0x00; // FEATURES (bit 0 = TRIM[1])
cdb[4] = (count >> 8) & 0xFF;   // COUNT (15:8)
    cdb[5] = count & 0xFF;          // COUNT (7:0)
    cdb[6] = (lba >> 24) & 0xFF;    // LBA (31:24)
    cdb[7] = lba & 0xFF;            // LBA (7:0)
    cdb[8] = (lba >> 32) & 0xFF;    // LBA (39:32)
    cdb[9] = (lba >> 40) & 0xFF;    // LBA (47:40)
    cdb[10] = (lba >> 16) & 0xFF;   // LBA (23:16)
    cdb[11] = 0x00; // DEVICE
    cdb[12] = 0x07; // COMMAND = DATA SET MANAGEMENT XL
    cdb[13] = 0x00;

    ret = scsi_command(fd, SG_DXFER_TO_DEV, cdb, sizeof(cdb), buffer, buffer_sz, sense, sizeof(sense));
    free(buffer);
    return ret;
}

/**
* Query device to determine if DATA SET MANAGEMENT XL
* command is supported
* parameters. It shall read log address 0x30, log page
* 3, and parse QWord 1,
* * bit 50.
* * fd          Send command to this file desciption
* * returns:
*   * supported   1 if supported, 0 if not supported
* *
*/
uint8_t is_dsm_xl_supported(int fd) {
    uint8_t supported = 0;
    uint8_t buffer[ATA_SECTOR_SIZE] = { 0 };

uint64_t supported_capabilities;

    if (!ata_read_log_ext(fd, 0x30, 3, buffer, ATA_SECTOR_SIZE)) {
        supported_capabilities = array_to_le_u64(1, buffer, ATA_SECTOR_SIZE);
        supported = (supported_capabilities >> 50) & 1;
    }
    return supported;
}

/**
* Trim entire LBA range of device path
* parameters
*  data        device_data_t structure
* return
*  error       0 upon success
*              1 if DSM XL trim command fails
*              2 if path could not be opened
*              -1 if DSM XL is not supported
*/
int trim_entire_drive(struct device_data_t *data) {
    dsm_trim_xl_range_entry entry[1];
    int ret = 0;
    int fd;
    int i;
    if (data->supported) {
        fd = open(data->path, O_RDONLY);
        if (fd < 0) {
            fprintf(stderr, "failed to open %s
", data->path);
            return 2;
        }
        entry[0].lba_value = 0;
        entry[0].range_length = data->total_lbas;
        printf("Trim XL on %s\n", data->path);
        printf("START LBA RANGE LENGTH\n");
        printf("------------- --------------\n");
        for (i = 0; i < sizeof(entry) / sizeof(entry[0]));
            printf("0x%012lx 0x%012lx\n", entry[i].lba_value, entry[i].range_length);
        i++;
            printf("0x%012lx 0x%012lx\n", entry[i].lba_value, entry[i].range_length);
        }
        if (!ata_trim_xl(fd, entry, sizeof(entry) / sizeof(entry[0]))) {
            printf("DSM Trim XL ok.\n");
        } else {
            ret = 1;
        }
    close(fd);
    } else {
        printf("DSM Trim XL is not supported.\n");
    // serial number at word 10, 20 bytes
    memcpy(serial, id_buffer + 10 * 2, 20);
    serial[20] = 0;
    byteswap(serial);
    // model number at word 27, 40 bytes
    memcpy(model, id_buffer + 27 * 2, 40);
    model[40] = 0;
    byteswap(model);
    trim(model);
    // total LBAs at word 100, 8 bytes
    lbas = array_to_le_u64(0, id_buffer + 100 * 2, 8);
    printf("Path: %s\n", path);
    return ret;
}
```c
printf("Model: %s\n", model);
printf("Serial: %s\n", serial);
printf("LBA: %lu\n", lbas);

supported = is_dsm_xl_supported(fd);
close(fd);

found = 0;
for (i = 0; i < _device_count; i++) {
    if (strcmp(_devices[i].path, path) == 0) {
        found = 1;
        _devices[i].supported = supported;
        _devices[i].total_lbas = lbas;
        break;
    }
}
if (!found) {
    // add it, if and only if there's enough space
    if (_device_count < MAX_DEVICES) {
        memset(_devices[_device_count].path, 0,
FILENAME_MAX);
        strcpy(_devices[_device_count].path, path,
FILENAME_MAX - 1);
        _devices[_device_count].supported =
        supported;
        _devices[_device_count].total_lbas =
        lbas;
        ++_device_count;
    } else {
        fprintf(stderr, "too many devices (max is
    %u)\n", MAX_DEVICES);
        return 3;
    }
    return 0;
}

/**
 * Trim all LBA on one device (passed in via command
line argument).
 */
int main(int argc, char* argv[]) {
    int ret;
    if (argc < 2) {
        fprintf(stderr, "missing device path\n\n");
        return 128;
    }
```