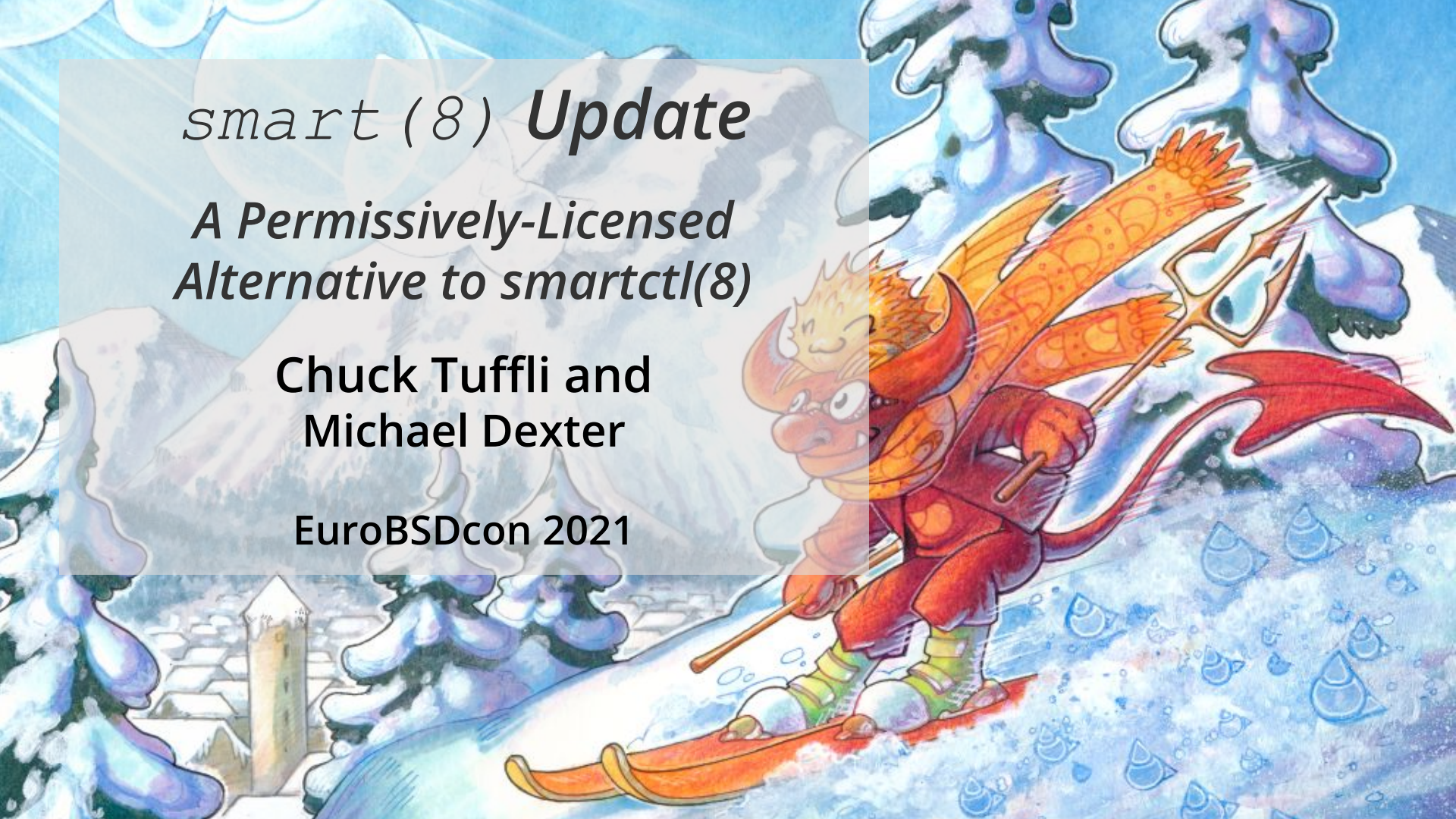


`smart(8)` *Update*

A Permissively-Licensed Alternative to smartctl(8)

Chuck Tuffli and
Michael Dexter

EuroBSDcon 2021

`smart(8)` Genesis

- *“Let’s do to smartctl what mandoc did to groff”*
- Formally proposed as `diskctl(8)` in a 2016 AsiaBSDCon paper by Michael
- Inspired by OpenBSD/NetBSD `atactl(8)`, hence the name
- First prototyped (poorly) with camcontrol

```
camcontrol cmd ada0 -a "B0 D0 00 4F C2 00 00 00 00 00 00 00" \
-i 512 - | od -tx1
0000000 01 00 05 33 00 64 64 00 00 00 00 00 00 00 09 32
0000020 00 63 63 63 02 00 00 00 00 00 0c 32 00 63 63 49...
```

`smart(8)` Design Goals

- *"The Plural of Regex is Regrets"* – `smartctl` output is neither human nor machine-readable, though it now supports JSON output
- At a minimum, unambiguous tab-separated values for save and easy scripting
- Extensible output format (text, json, xml, ...) within reason
- Modularity with a portable library for use in say, OpenZFS
- Possibility of OpenZFS syntax... `smart -o 5,196,197`
- Possibility of exposure via `sysctls`
- ISC, BSD or MIT-licensed for universal compatibility

`smart(8)` *"Hey, that sounds interesting"*

- Chuck had attempted to port NVMe's SMART to an ATA-oriented application
  - ATA vs. NVMe health reporting data elements vary *wildly*
  - !@#$ "I don't have time for this" → PUNT
- Is `libsmart` possible?
  - `smart(8)` could provide the test platform
  - https://github.com/ctuffli/smart (mirror)
  - FreeBSD ports/pkg : `sysutils/smart`

S.M.A.R.T. Crash Course

- Brief History
- Your devices to *not* output anything near what you see in `smartctl`
- “S.M.A.R.T.” data is either a list, structure, or log pages of numerical values
- Vendors do not agree on the log page values
- A Venn diagram between ATA, SCSI, and NVMe is mostly possible

ATA != NVMe != SCSI

ATA != NVMe != SCSI

- Each protocol retrieves disk health differently
  - ATA : SMART Read Data command
  - NVMe : SMART/Health Information log page
  - SCSI/SAS : Write/Read/Verify/Non-Medium/Last N Error, ... log pages

ATA != NVMe != SCSI

- Content mostly different. Sort of.
  - ATA : Write Error Rate
  - NVMe : Media and Data Integrity Errors
  - SCSI : Write Total uncorrected errors

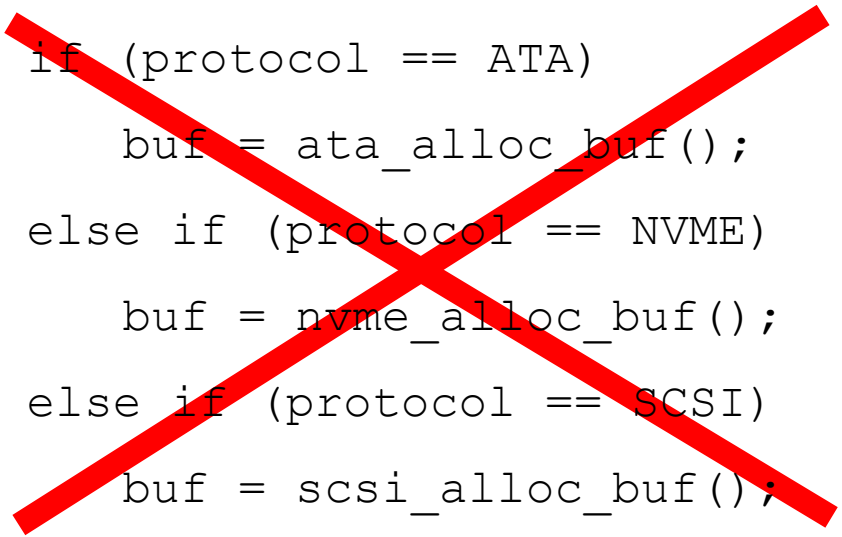
ATA != NVMe != SCSI

- Standards-based vs. ... not
- NVMe and SCSI : Content of log page(s) defined by standards groups
  - NVM Express Technical Working Group
  - T10
- ATA : no standard(*) / each vendor allowed to
  - decided which attribute ID's to support
  - decide what the attribute ID means

What would I want?

```
if (protocol == ATA)
    buf = ata_alloc_buf();
else if (protocol == NVME)
    buf = nvme_alloc_buf();
else if (protocol == SCSI)
    buf = scsi_alloc_buf();
```

What would I want?



```
if (protocol == ATA)
    buf = ata_alloc_buf();
else if (protocol == NVME)
    buf = nvme_alloc_buf();
else if (protocol == SCSI)
    buf = scsi_alloc_buf();
```

What would I want?

```
if (protocol == ATA)
    buf = ata_alloc_buf();
else if (protocol == NVME)
    buf = nvme_alloc_buf();
else if (protocol == SCSI)
    buf = scsi_alloc_buf();
```

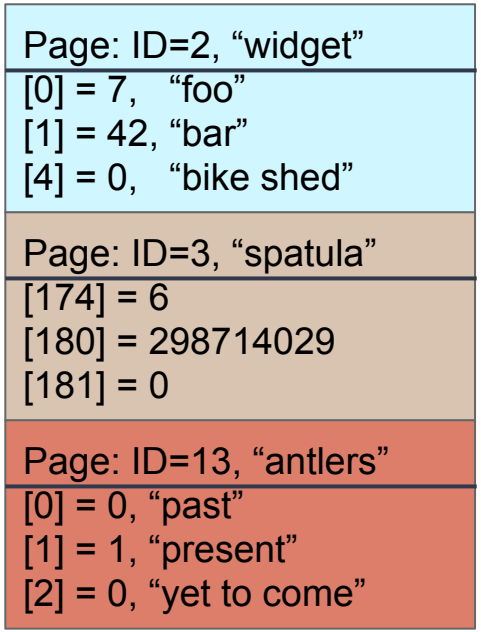
- Protocol independent structure for data (“DUMB”)
- Self-describing buffer (“Maps”)
- OS dependent / independent split

Dumb Unified Model for smart Buffers (“DUMB”)

	ATA	NVMe	SCSI
How	SMART Read Data command	SMART/Health Information Log page	Write Errors log page Read Errors log page Verify Errors log page Non-medium Errors log page Last N Errors log page Temperature log page Start-Stop Cycle Count log page Informational Exceptions log page
Elements	List of attribute ID’s	Packed data structure	“Parameter Codes” + packed data structure
Data	Raw value, flags, min, max, threshold (8-48 bits)	Number (1-128 bits)	Numbers (big endian), strings (!@#$) Variable size

Dumb Unified Model for smart Buffers (“DUMB”)

- Object Model for Health Data
  - List of log pages
  - Page is list of unique attribute ID’s
  - Attributes have a value, description, size, etc.
- "That looks like SCSI"



Object Model Mapping – ATA

- Doesn't have log pages. Use value from Command Feature field
  - E.g. SMART READ DATA (`0xd0` or `208` decimal)
- Use Attribute IDs (unique)
- Use returned raw value and divining rod for description(*)

Object Model Mapping – ATA

- Doesn't have log pages. Use value from Command Feature field
  - E.g. SMART READ DATA (`0xd0` or `208` decimal)
- Use Attribute IDs (unique)
- Use returned raw value and divining rod for description(*)

```
# smart ada0 | head -3
208  5    0
208  12   73
208  175  0
```

Object Model Mapping – NVMe

- Use SMART/Health Information log page ID (`0x2`)
- Use byte offset of each field as the attribute ID (unique)
- Use value / description as defined by NVMe specification

Object Model Mapping – NVMe

- Use SMART/Health Information log page ID (`0x2`)
- Use byte offset of each field as the attribute ID (unique)
- Use value / description as defined by NVMe specification

```
# smart nda0 | head -3
2    0    0
2    1    309
2    3    100
```

Figure 207: Get Log Page – SMART / Health Information Log

Bytes	Description
00	**Critical Warning**: This field indicates critical warnings for the state of the controller.
02:01	**Composite Temperature**: Contains a value corresponding to a temperature in Kelvins
03	**Available Spare**: Contains a normalized percentage (0% to 100%) of the remaining spare capacity available.

Object Model Mapping – NVMe

- Use SMART/Health log page ID (`0x2`)
- Use byte offset of each field as the attribute ID (unique)
- Use value / description as defined by NVMe specification

Needs the CAM NVMe driver!

`hw.nvme.use_nvd=0`

```
# smart nda0 | head -3
2    0    0
2    1    309
2    3    100
```

Figure 207: Get Log Page – SMART / Health Information Log

Bytes	Description
00	**Critical Warning**: This field indicates critical warnings for the state of the controller.
02:01	**Composite Temperature**: Contains a value corresponding to a temperature in Kelvins
03	**Available Spare**: Contains a normalized percentage (0% to 100%) of the remaining spare capacity available.

Object Model Mapping – SCSI

- Use log page ID
- Use parameter code as the attribute ID (**not unique across pages!**)
- Use value / description as defined by SCSI specification

Object Model Mapping – SCSI

- Use log page ID
- Use parameter code as the attribute ID (**not unique across pages!**)
- Use value / description as defined by SCSI specification

```
# smart da0
2    0    0
2    1    108277
5    0    0
5    1    0
6    0    0
13   0    31
13   1    85
```

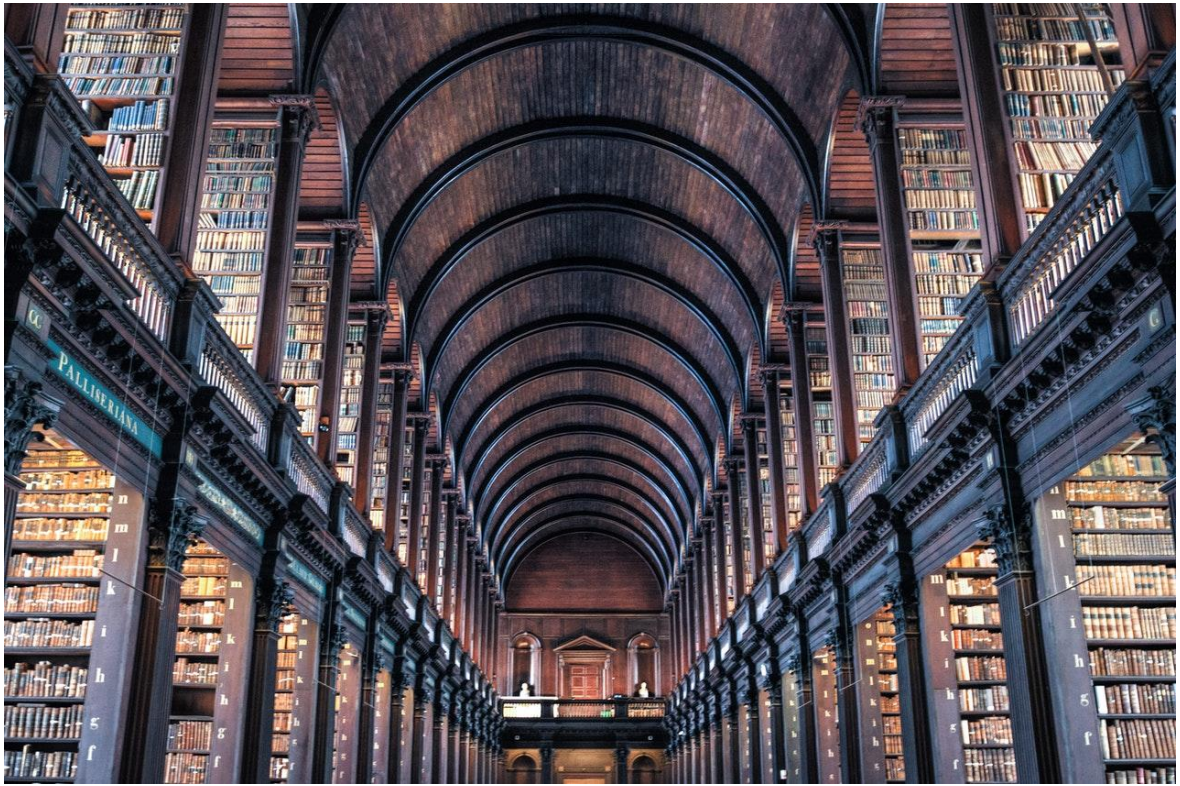
Table 264 Log page codes

Page Code	Log Page Name
02h	Write Error Counter
05h	Verify Error Counter
06h	Non-Medium Error
0Dh	Temperature

Table 351 Temperature log page parameter codes

Parameter Code	Description
0000h	Temperature (℃)
0001h	Reference Temperature (℃)

libsmart



“Handle”

- Abstract connection to lower “gunk”
- Device independent / dependent
- Allocated by device layer

```
typedef void * smart_h;
```

Application

```
typedef struct smart_s {
    smart_protocol_e protocol;
    smart_info_t info;
    smart_page_list_t *pg_list;
} smart_t;
```

Device independent (library)

```
struct fbsd_smart {
    smart_t common;
    struct cam_device *camdev;
};
```

Device dependent

“Maps”

- Self-describing buffer of attributes
  - Buffer returned by device
  - Number of attributes
  - Array of attributes

```
typedef struct smart_map_s {
    smart_buf_t *sb;                /* Protocol, OS/device buffer, size , count */
    uint32_t count;                 /* Number of attributes */
    smart_attr_t attr[];            /* Array of attributes */
} smart_map_t;
```

“Attributes”

- Identifier tuple (page + id)
- Pointer to data
  - Size of data (number of bytes)
  - Flags (big endian, data is a string, ...)

```
typedef struct smart_attr_s {
    uint32_t page;
    uint32_t id;
    char *description;              /* human readable description */
    uint32_t bytes;
    uint32_t flags;
    void *raw;
    struct smart_map_s *thresh;     /* Threshold values (if any) */
} smart_attr_t;
```

Device / OS Abstraction

- Currently for FreeBSD
  - Storage device interface (aka “CAM”) made this easy
- Did a PoC for Windows
  - via openSeaChest (https://github.com/Seagate/openSeaChest)

API	Description
open	Open a device to gather SMART information
close	Close a device and release the associated resources
read_log	Read the log page

Library – libsmart

API	Description
open	open the specified device + return “handle”
close	Close the device
supported	Does the device support health data?
read	Read health data from device and create a “map”
free	Deallocate memory used for “map”
print	Print health data
print_device_info	Print device information (vendor, device, revision)

Application – smart

- Option processing
- libxo setup / teardown
- SMART library open, print, free, close

```
Usage: smart [-htxidDv] [-a <attribute id>] <device name>
        -h, --help
        -t, --threshold : also print out the threshold
values
        -x, --hex : print the values out in hexadecimal
        -a, --attribute : print a specific attribute
        -i, --info : print general device information
        -d, --decode: decode the attribute IDs
        -D, --no-decode: don't decode the attribute IDs
        -v, --version : print the version and copyright
            --debug : output diagnostic information
```

Output Format

- Original motivation, collect values over time
  - Know specific attribute(s)
  - Only need raw value
  - Called via cron(8) or monitoring framework (e.g. Prometheus)

```
# smart --attribute 5 ada0
0
```

Output Format

- Driven by libxo

  libxo – A Library for Generating Text, XML, JSON, and HTML Output

```
# smart --attribute 5 --libxo=json,pretty ada0
{
  "drive": {
    "attributes": {
      "attribute": [
        {
          "raw": 0
        }
      ]
    }
  }
}
```

Output Format


```
{
  "drive": {
    "device": "SAMSUNG SSD PM871 M.2 2280 256GB",
    "rev": "SAMSUNG SSD PM871 M.2 2280 256GB",
    "serial": "S208NXAGA03210",
    "attributes": {
      "attribute": [
        {
          "page": 208,
          "id": 5,
          "raw": 0
        },
        ...
      ]
    }
  }
}
```

Output Format

- By popular demand, attribute decode
  - SCSI : text from the specification
  - NVMe : text from the specification, filtered by version
  - ATA : it’s complicated

```
# smart --decode ada0
Reallocated Sectors Count       5       0
Power Cycle Count       12      74
Power Loss Protection Failure   175     0
Erase Fail Count (chip) 176     0
Wear Range Delta        177     11
208     178     0
Used Reserved Block Count Total 179     0
208     180     771
```

Decoding ATA – smartmon

- SMART attribute structure specified, but not attributes themselves
- Get definitions from each vendor
- Use regex on drive model + firmware revision

```
"WDC WD(7500BFCX|10JFCX|[1-6]0EFRX|[68]0E[FZ]ZX|(8|10)0EFAX|120EMFZ)-.*"
```

  - smartmontools
  - Drive database (drivedb.h)
    - 6200 LoC, 28+KBytes
    - GPL

Decoding ATA – smart

- ANSI - INCITS TR-54 – “SMART Attribute Descriptions (SAD)”
- Documents agreed upon definitions
- Future: include text-based drive database

```
# smart --decode ada0
Reallocated Sectors Count       5       0
Power Cycle Count       12      74
Power Loss Protection Failure   175     0
Erase Fail Count (chip) 176     0
Wear Range Delta        177     11
208     178     0
Used Reserved Block Count Total 179     0
208     180     771
```

Not all ID’s agreed upon 😿

Thank you!

- https://foss.heptapod.net/bsdutils/smart (development)
- https://github.com/ctuffli/smart (mirror)
- FreeBSD ports/pkg : `sysutils/smart`
- Contact us

  chuck@tuffli.net / @ctuffli

  editor@callfortesting.org / @michaeldexter
- Questions?