

Phase 2 – Defining a Standard for SMART Attribute Labels

1. Background

In Phase 1, SSD endurance metric parsing in `smartmontools` was standardized by:

- Expanding regex coverage for vendor-specific lifetime/wear attributes.
- Normalizing “remaining” values into a unified `% used` metric.

Producing a single JSON key for consistency:

```
"endurance_used": { "current_percent": X }
```

This improved runtime consistency but did not address the **lack of a universal naming convention** for SMART attributes. Phase 2 focuses on defining such a convention so future attributes are consistent from the start.

2. Problem

- Attribute names in `drivedb.h` are inconsistent and vendor-specific.
 - No explicit indication of whether a value is “% used” or “% remaining.”
 - Temperature and LBAs written metrics have the same inconsistency issue.
 - Monitoring tools like Prometheus struggle to aggregate values across a fleet without custom logic.
-

3. Goals

1. Define a **uniform naming standard** for key metrics:
 - SSD lifetime used
 - SSD lifetime remaining
 - Drive temperature
 - LBAs written
 2. Provide a Markdown document specifying the rules for naming new attributes.
 3. Ensure new `drivedb.h` entries follow the standard so monitoring tools can use them directly.
-

4. Proposed Standard Labels

Metric Type	Standard Label (Used)	Standard Label (Remaining)	Notes
SSD Lifetime	<code>SSD_Lifetime_Used_Pct</code>	<code>SSD_Lifetime_Remain_Pct</code>	Always suffix with <code>_Pct</code> for percentages.
LBAs Written	<code>Total_LBAs_Written</code>	—	Raw count of LBAs; no remaining variant.
Temperature	<code>Temperature_Celsius</code>	—	Always in Celsius; raw field is numeric.

Workload Wear	<code>Workload_Media_Wear _Pct</code>	—	Optional; for drives that expose workload wear separately.
--------------------------	-------------------------------------------	---	---------------------------------------------------------------------

Polarity Rule:

- Labels must clearly indicate whether they represent *used* or *remaining* percentages.
- This removes ambiguity and the need for runtime inversion logic in most cases.

5. Implementation Plan

Step 1 – Documentation

Create a file named `ATTRIBUTE_LABEL_STANDARD.md` containing:

- Problem statement (inconsistent naming, polarity confusion).
- Full list of approved attribute names.
- Rules for creating new labels:
 - Consistent casing (`Pascal_Case` with underscores).
 - Units explicitly stated in the name (`_Pct`, `_GiB`).
 - No `%` symbol in the literal attribute name.

Step 2 – Runtime Integration

- Keep Phase 1 regex normalization in `ataprtint.cpp` for compatibility with older `drivedb.h` entries.
- Prefer direct parsing of standard labels when present in SMART data.

Step 3 – Upstream PR

- Submit a PR adding `ATTRIBUTE_LABEL_STANDARD.md` to the smartmontools repository.
 - Include examples showing how standardized labels improve cross-vendor monitoring consistency.
-

6. Improvements Over Phase 1

- Eliminates the root cause of metric inconsistency for future attributes.
 - Provides explicit naming rules to avoid vendor quirks in new labels.
 - Reduces reliance on complex regex matching over time.
 - Maintains focus on **enterprise SSDs** for predictable, documented behavior.
-

7. Expected Outcome

- A published naming standard that future `drivedb.h` entries will follow.
- Easier fleet-wide analytics with Prometheus, Grafana, and other monitoring tools.
- Gradual simplification of parsing logic as standardized attributes replace legacy variants.

Here's a draft for `ATTRIBUTE_LABEL_STANDARD.md` file for Phase 2:

SMART Attribute Label Standard

1. Purpose

This document defines a standard naming convention for SMART attribute text labels in `drivedb.h`.

The goal is to:

- Eliminate inconsistent and vendor-specific attribute names.
- Make metric polarity (*used* vs *remaining*) explicit.
- Simplify integration with monitoring systems such as Prometheus and Grafana.
- Ensure future attributes can be queried consistently across all drives in a fleet.

2. Problems with Current Labels

- **Inconsistent naming:** Multiple labels exist for the same metric (e.g., `SSD_Life_Left`, `Percent_Lifetime_Remain`).
- **Polarity ambiguity:** Some metrics represent “% used”, others “% remaining” — often without indication in the label.
- **Unit confusion:** Labels sometimes include symbols like % instead of indicating units in the name.
- **Vendor quirks:** Some names contain vendor-specific wording that doesn't generalize.

3. General Naming Rules

1. Case & Formatting:

- Use `Pascal_Case` with underscores. Example: `SSD_Lifetime_Used_Pct`.

2. Polarity:

- Include `_Used` or `_Remain` in the label to indicate metric polarity.

3. Units:

- Percentages → suffix `_Pct`
- Capacities in bytes → suffix `_Bytes`
- Capacities in gibibytes → suffix `_GiB`
- Counts with no units → no suffix required
- Never include symbols like `%` or `°C` directly in the label text.

4. Clarity:

- Avoid vendor-specific wording unless unavoidable.
- Avoid abbreviations that are not industry-standard.

5. Uniqueness:

- Each attribute name must be unique in the database.

4. Standard Attribute Labels

4.1 SSD Lifetime Metrics

Metric Purpose	Standard Label	Unit
Lifetime used (%)	<code>SSD_Lifetime_Used_Pct</code>	Percent

Lifetime remaining (%)	SSD_Lifetime_Remain_Pct	Percent
------------------------	-------------------------	---------

4.2 LBAs Written

Metric Purpose	Standard Label	Unit
Total LBAs written	Total_LBAs_Written	Count (blocks)

4.3 Temperature

Metric Purpose	Standard Label	Unit
Drive temperature	Temperature_Celsius	Degrees Celsius

4.4 Optional / Vendor-Specific Wear Metrics

Metric Purpose	Standard Label	Unit
Workload media wear (%)	Workload_Media_Wear_Pct	Percent

5. Examples of Conversions

When adding or updating `drivedb.h`, replace vendor-specific or inconsistent names with the standard labels:

Vendor Label	Standard Label
<code>Percent_Lifetime_Remain</code>	<code>SSD_Lifetime_Remain_Pct</code>
<code>SSD_Life_Left</code>	<code>SSD_Lifetime_Remain_Pct</code>
<code>DriveLife_Used%</code>	<code>SSD_Lifetime_Used_Pct</code>
<code>Lifetime_Writes_GiB</code>	<code>Total_LBAs_Written</code>
<code>Temperature_Internal</code>	<code>Temperature_Celsius</code>

6. Enforcement

- All **new** `drivedb.h` entries **must** comply with these rules.
 - PRs introducing non-standard labels will be requested to change them before merge.
 - Backward compatibility for existing drives will be maintained via runtime regex mapping until older entries are updated.
-

7. Benefits

- Eliminates the need for complex regex parsing for new attributes.
- Makes metric polarity explicit, reducing interpretation errors.
- Enables uniform queries across different vendors in monitoring systems.
- Improves maintainability and readability of `drivedb.h`.

Phase 1 Improvements Considered:

SSD Endurance Metrics – Before vs After Changes

Aspect	Before Changes	After Changes
Regex Coverage	Limited <code>endurance_regex</code> with a few common names like <code>SSD_Life_Left</code> , <code>Percent_Life_Used</code> . Missed many vendor variations and truncated forms.	Expanded <code>endurance_regex</code> to match a comprehensive set of endurance/lifetime attributes from <code>drivedb.h</code> , including case-insensitive variants, optional suffixes (e.g., <code>,SSD</code>), and percentage-based spare-life metrics.
Remaining Life Detection	No dedicated detection — relied on hardcoded checks for a few attributes.	Introduced <code>remaining_regex</code> to automatically detect and invert “remaining life” values into “% used”. Tightened patterns to avoid false positives.
Attribute Scope	Mixed approach, sometimes included ambiguous or raw counters.	Clearly classified attributes into in scope (direct endurance metrics), maybe , and out of scope (e.g., <code>Life_Curve_Status</code> , raw spare block counts).

Normalization Logic	No unified calculation; each attribute had to be handled separately.	Unified logic: If match in <code>remaining_regex</code> , <code>endurance_used = 100 - normval</code> ; else, <code>endurance_used = normval</code> . Stored in JSON as <code>endurance_used.current_percent</code> .
Output Consistency	Different drives exposed different attribute names; no single JSON key for endurance.	All matched endurance attributes are normalized into a single consistent JSON field : <code>json { "endurance_used": { "current_percent": X } }</code>
Testing	Informal or attribute-specific tests only.	Two-stage testing: 1. Regex unit test (<code>test_regex.cpp</code>) for correct matches. 2. Live SSD test with patched <code>smartctl</code> confirming JSON output for <code>endurance_used</code> , <code>temperature_celsius</code> , and <code>lbas_written</code> .
Focus	No explicit priority on enterprise vs client drives.	Focused on enterprise SSD metrics for consistent large-scale monitoring; excluded poorly documented client-specific attributes.

This table makes it very clear how I went from a **partial, hardcoded** approach to a **comprehensive, automated, and standardized** endurance metric parsing system.

```
// --- Regex for all endurance/lifetime-related SMART attributes ---
static const regular_expression endurance_regex(
    "(?i)" // case-insensitive
    "SSD_Life_Left(?:\\(0\\.01%\\))?.*|"
    "SSD_LifeLeft.*|"
    "DriveLife_(?:Used%|Remaining%).*|"
    "Drive_Life_(?:Used%|Remaining%).*|"
    "Percent_Life_(?:Used|Remaining%).*|"
```

```

    "Percent_Lifetime_(?:Used|Remain).*"
    "PCT_Life_Remaining.*|"
    "Perc_Rated_Life_(?:Used|Remain).*"
    "Remaining_Life.*|"
    "Life_Remaining.*|"
    "Lifetime_Remaining%.*|"
    "Lifetime_Left.*|"
    "SSD_Remaining_Life_Perc.*|"
    "Sys_Percent_Life_Remain.*|"
    "Wear_Leveling(?:_Count)?.*|"
    "Media_Wearout_Indicator.*|"
    "Workld_Media_Wear_Indic.*|"
    "End_of_Life.*"
);

// --- Regex to detect "remaining life" semantics ---
static const regular_expression remaining_regex(
    "(?i)"
    ".*Remaining.*|"
    ".*Remain.*|"
    ".*Left.*|"
    ".*End_of_Life.*"
);

// --- Regex for temperature attributes ---
static const regular_expression temperature_regex(
    "(?i)"
    ".*Temperature.*|.*Temp.*"
);

// --- Regex for LBA written / NAND write attributes ---
static const regular_expression lba_written_regex(
    "(?i)"
    ".*LBAs.*Written.*|"
    ".*Host.*Writes.*|"
    ".*Writes.*GiB.*|"
    ".*Total.*Writes.*|"
    ".*NAND.*Writes.*|"
    ".*Program_Page_Count.*"
);

// --- SMART attribute parsing ---
if (endurance_regex.full_match(name)) {

```

```

bool isRemaining = remaining_regex.full_match(name);
double endurance_percent;

if (isRemaining) {
    // Invert for "remaining life" attributes -- only if within 0-100
    if (normval >= 0 && normval <= 100) {
        endurance_percent = 100.0 - normval;
    } else {
        // Out-of-range values (e.g., Apple SSD quirk: 168%) -- keep
raw
        endurance_percent = normval;
    }
} else {
    endurance_percent = normval;
}

jglb["endurance_used"]["current_percent"] = endurance_percent;
return;
}

if (temperature_regex.full_match(name)) {
    jglb["temperature_celsius"] = normval;
    return;
}

if (lba_written_regex.full_match(name)) {
    // Store raw -- unit conversion can be added at JSON consumer side
    jglb["lbas_written"] = normval;
    return;
}

// Debug logging only for unmatched attributes
#ifdef NDEBUG
std::cerr << "Unmatched SMART attribute: " << name
    << " (ID: " << id << ", normval: " << normval << ")"
    << std::endl;
#endif

```

<https://github.com/smartmontools/smartmontools/pull/378>

Draft Pull Request – Phase 1: SSD Endurance Normalization in smartmontools

Summary

This draft PR implements logic to detect and normalize SSD endurance-related SMART attributes in smartmontools. It introduces regex-based matching for vendor-specific attribute labels, classifies attributes as representing used or remaining endurance, and outputs a unified endurance metric in the JSON report.

Motivation

SSD vendors report endurance metrics under widely varying attribute names in `drivedb.h`, such as variations of "life used", "life remaining", "media wearout", and others. This inconsistency complicates meaningful comparison of drive health, especially when monitoring large fleets using tools like Prometheus exporters.

This PR addresses the following issues:

- Incomplete regex matching in `ataprint.cpp` for endurance-related attributes.
- Lack of normalization between attributes reporting % used and % remaining endurance.

- Absence of a consistent "endurance_used" metric in the JSON output.

Key Changes

* **Expanded endurance_regex and remaining_regex:**

Covers a broader range of vendor-specific endurance labels such as SSD_Life_Left, Wear_Leveling_Count, Percent_Lifetime_Remain, and others.

* **Normalization Logic:**

* If the attribute indicates *remaining* life, the value is inverted: $100 - x$.

* Otherwise, the raw normalized value is used directly.

* The normalized value is stored under `jglb["endurance_used"]["current_percent"]` in the JSON output.

* **Improved Regex Coverage:**

Patterns mined from `drivedb.h` were added to capture less common labels like `Lifetime_Remaining%`, `End_of_Life`, etc.

Execution Flow

The endurance normalization logic is summarized in the following flowcharts:

Figure 1: Attribute matching and inversion logic.

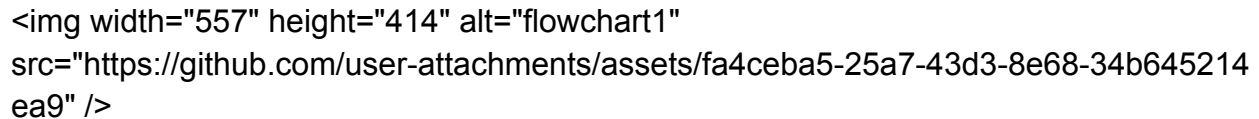
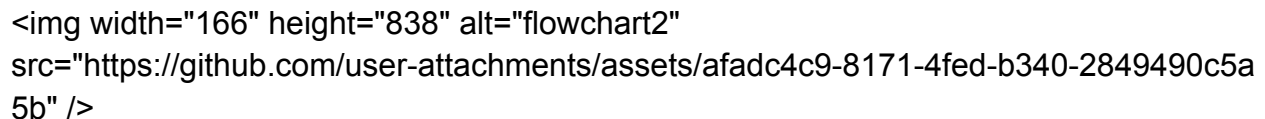
A flowchart illustrating the attribute matching and inversion logic. The flowchart starts with a decision point for attribute matching, leading to various processing steps and loops.

Figure 2: Overall parsing and output flow.

A flowchart illustrating the overall parsing and output flow. The flowchart starts with a decision point for parsing, leading to various processing steps and loops.

Changes in `src/ataprint.cpp`

For each SMART attribute `(id, name, normval)` processed, the following checks and actions are performed:

1. Endurance Check

* If the attribute name matches `endurance_regex`:

- * If it also matches ``remaining_regex``, calculate endurance as:

- ``endurance_used.current_percent = 100 - normval``

- * Else, use the normalized value directly:

- ``endurance_used.current_percent = normval``

- * Exit the attribute check early after storing the endurance value.

2. Temperature Check

- * If the attribute name matches ``temperature_regex``, store the value as:

- ``temperature_celsius = normval``

3. LBA Written Check

- * If the attribute name matches ``lba_written_regex``, store the value as:

- ``lba_written = normval``

4. Other Attributes

- * For attributes that do not match any of the above regex patterns:

- * Log the unmatched attribute for debugging purposes (enabled only in debug builds).

****Success Path:**** JSON keys populated for matched attributes.

****Failure Path:**** Attributes unmatched — only logged in debug.

Sample Test Cases

Example Output

...

json

```
"endurance_used": {  
  "current_percent": 76  
}
```

...

Internals Touched

* ataprint.cpp:

* Regex definitions for endurance and remaining life attributes.

* Normalization logic implementation.

* Conditional parsing extended for endurance, temperature, and LBAs written.

Notes

- This PR is in draft so regex coverage can be refined with more drive samples.

- macOS testing has limitations for Apple NVMe SSDs due to restricted SMART visibility.

- Future work: handle more edge-case labels (e.g., Reserve_Blks_Remaining).

To Do (Phase 2 / Final PR)

* Add fuzzy matching to capture vendor-specific variations not covered by static regex.

* Extend parsing logic to handle ambiguous attributes like Life_Curve_Status.

* Improve testing coverage on Linux (Ubuntu) across SATA and NVMe SSDs.

* Finalize PR by cleaning up commits and adding documentation comments.