Naming and data format standards for sysfs files

The libsensors library offers an interface to the raw sensors data through the sysfs interface. Since lm-sensors 3.0.0, libsensors is completely chip-independent. It assumes that all the kernel drivers implement the standard sysfs interface described in this document. This makes adding or updating support for any given chip very easy, as libsensors, and applications using it, do not need to be modified. This is a major improvement compared to lm-sensors 2.

Note that motherboards vary widely in the connections to sensor chips. There is no standard that ensures, for example, that the second temperature sensor is connected to the CPU, or that the second fan is on the CPU. Also, some values reported by the chips need some computation before they make full sense. For example, most chips can only measure voltages between 0 and +4V. Other voltages are scaled back into that range using external resistors. Since the values of these resistors can change from motherboard to motherboard, the conversions cannot be hard coded into the driver and have to be done in user space.

For this reason, even if we aim at a chip-independent libsensors, it will still require a configuration file (e.g. /etc/sensors.conf) for proper values conversion, labeling of inputs and hiding of unused inputs.

An alternative method that some programs use is to access the sysfs files directly. This document briefly describes the standards that the drivers follow, so that an application program can scan for entries and access this data in a simple and consistent way. That said, such programs will have to implement conversion, labeling and hiding of inputs. For this reason, it is still not recommended to bypass the library.

Each chip gets its own directory in the sysfs /sys/devices tree. To find all sensor chips, it is easier to follow the device symlinks from */sys/class/hwmon/hwmon**.

Up to lm-sensors 3.0.0, libsensors looks for hardware monitoring attributes in the "physical" device directory. Since lm-sensors 3.0.1, attributes found in the hwmon "class" device directory are also supported. Complex drivers (e.g. drivers for multifunction chips) may want to use this possibility to avoid namespace pollution. The only drawback will be that older versions of libsensors won't support the driver in question.

All sysfs values are fixed point numbers.

There is only one value per file, unlike the older /proc specification. The common scheme for files naming is: <type> <number>_<item>. Usual types for sensor chips are "in" (voltage), "temp" (temperature) and "fan" (fan). Usual items are "input" (measured value), "max" (high threshold, "min" (low threshold). Numbering usually starts from 1, except for voltages which start from 0 (because most data sheets use this). A number is always used for elements that can be present more than once, even if there is a single element of the given type on the specific chip. Other files do not refer to a specific element, so they have a simple name, and no number.

Alarms are direct indications read from the chips. The drivers do NOT make comparisons of readings to thresholds. This allows violations between readings to be caught and alarmed. The exact definition of an alarm (for example, whether a threshold must be met or must be exceeded to cause an alarm) is chip-dependent.

When setting values of hwmon sysfs attributes, the string representation of the desired value must be written, note that strings which are not a number are interpreted as 0! For more on how written strings are interpreted see the "sysfs attribute writes interpretation" section at the end of this file.

Attribute access

Hardware monitoring sysfs attributes are displayed by unrestricted userspace applications. For this reason, all standard ABI attributes shall be world readable. Writeable standard ABI attributes shall be writeable only for privileged users.

[0-]*	denotes any positive number starting from 0
[1-]*	denotes any positive number starting from 1
RO	read only value
WO	write only value
RW	read/write value

Read/write values may be read-only for some chips, depending on the hardware implementation.

All entries (except name) are optional, and should only be created in a given driver if the chip has the feature.

See Documentation/ABI/testing/sysfs-class-hwmon for a complete description of the attributes.

Global attributes

name

The chip name.

label
: A descriptive label that allows to uniquely identify a device within the system.

update_interval
: The interval at which the chip will update readings.

Voltages

in[0-]_min*
: Voltage min value.

in[0-]_lcrit*
: Voltage critical min value.

in[0-]_max*
: Voltage max value.

in[0-]_crit*
: Voltage critical max value.

in[0-]_input*
: Voltage input value.

in[0-]_average*
: Average voltage

in[0-]_lowest*
: Historical minimum voltage

in[0-]_highest*
: Historical maximum voltage

in[0-]_reset_history*
: Reset inX_lowest and inX_highest

in_reset_history
: Reset inX_lowest and inX_highest for all sensors

in[0-]_label*
: Suggested voltage channel label.

in[0-]_enable*
: Enable or disable the sensors.

cpu[0-]_vid*
: CPU core reference voltage.

vrm
: Voltage Regulator Module version number.

in[0-]_rated_min*
: Minimum rated voltage.

in[0-]_rated_max*
: Maximum rated voltage.

Also see the Alarms section for status flags associated with voltages.

Fans

fan[1-]_min*
: Fan minimum value

fan[1-]_max*
: Fan maximum value

fan[1-]_input*
: Fan input value.

fan[1-]_div*
: Fan divisor.

fan[1-]_pulses*
: Number of tachometer pulses per fan revolution.

fan[1-]_target*
: Desired fan speed

fan[1-]_label*
: Suggested fan channel label.

fan[1-]_enable*
: Enable or disable the sensors.

Also see the Alarms section for status flags associated with fans.

PWM

pwm[1-]*
: Pulse width modulation fan control.

pwm[1-]_enable*

Fan speed control method:

pwm[1-]_mode*
: direct current or pulse-width modulation.

pwm[1-]_freq*
: Base PWM frequency in Hz.

pwm[1-]_auto_channels_temp*
: Select which temperature channels affect this PWM output in auto mode.

pwm[1-]_auto_point[1-*]_pwm / pwm[1-*]_auto_point[1-*]_temp / pwm[1-*]_auto_point[1-*]_temp_hyst*
: Define the PWM vs temperature curve.

temp[1-]_auto_point[1-*]_pwm / temp[1-*]_auto_point[1-*]_temp / temp[1-*]_auto_point[1-*]_temp_hyst*
: Define the PWM vs temperature curve.

There is a third case where trip points are associated to both PWM output channels and temperature channels: the PWM values are associated to PWM output channels while the temperature values are associated to temperature channels. In that case, the result is determined by the mapping between temperature inputs and PWM outputs. When several temperature inputs are mapped to a given PWM output, this leads to several candidate PWM values. The actual result is up to the chip, but in general the highest candidate value (fastest fan speed) wins.

Temperatures

temp[1-]_type*
: Sensor type selection.

temp[1-]_max*
: Temperature max value.

temp[1-]_min*
: Temperature min value.

temp[1-]_max_hyst*
: Temperature hysteresis value for max limit.

temp[1-]_min_hyst*
: Temperature hysteresis value for min limit.

temp[1-]_input*
: Temperature input value.

temp[1-]_crit*
: Temperature critical max value, typically greater than corresponding temp_max values.

temp[1-]_crit_hyst*
: Temperature hysteresis value for critical limit.

temp[1-]_emergency*
: Temperature emergency max value, for chips supporting more than two upper temperature limits.

temp[1-]_emergency_hyst*
: Temperature hysteresis value for emergency limit.

temp[1-]_lcrit*
: Temperature critical min value, typically lower than corresponding temp_min values.

temp[1-]_lcrit_hyst*
: Temperature hysteresis value for critical min limit.

temp[1-]_offset*
: Temperature offset which is added to the temperature reading by the chip.

temp[1-]_label*
: Suggested temperature channel label.

temp[1-]_lowest*
: Historical minimum temperature

temp[1-]_highest*
: Historical maximum temperature

temp[1-]_reset_history*
: Reset temp_lowest and temp_highest

temp_reset_history
: Reset temp_lowest and temp_highest for all sensors

temp[1-]_enable*
: Enable or disable the sensors.

temp[1-]_rated_min*
: Minimum rated temperature.

temp[1-]_rated_max*
: Maximum rated temperature.

Some chips measure temperature using external thermistors and an ADC, and report the temperature measurement as a voltage. Converting this voltage back to a temperature (or the other way around for limits) requires mathematical functions not available in the kernel, so the conversion must occur in user space. For these chips, all temp* files described above should contain values expressed in millivolt instead of millidegree Celsius. In other words, such temperature channels are handled as voltage channels by the driver.

Also see the Alarms section for status flags associated with temperatures.

Currents

curr[1-]_max*

Current max value.

curr[1-]_min*

Current min value.

curr[1-]_lcrit*

Current critical low value

curr[1-]_crit*

Current critical high value.

curr[1-]_input*

Current input value.

curr[1-]_average*

Average current use.

curr[1-]_lowest*

Historical minimum current.

curr[1-]_highest*

Historical maximum current.

curr[1-]_reset_history*

Reset currX_lowest and currX_highest

WO

curr_reset_history

Reset currX_lowest and currX_highest for all sensors.

curr[1-]_enable*

Enable or disable the sensors.

curr[1-]_rated_min*

Minimum rated current.

curr[1-]_rated_max*

Maximum rated current.

Also see the Alarms section for status flags associated with currents.

Power

power[1-]_average*

Average power use.

power[1-]_average_interval*

Power use averaging interval.

power[1-]_average_interval_max*

Maximum power use averaging interval.

power[1-]_average_interval_min*

Minimum power use averaging interval.

power[1-]_average_highest*

Historical average maximum power use

power[1-]_average_lowest*

Historical average minimum power use

power[1-]_average_max*

A poll notification is sent to *power[1-*]_average* when power use rises above this value.

power[1-]_average_min*

A poll notification is sent to *power[1-*]_average* when power use sinks below this value.

power[1-]_input*

Instantaneous power use.

power[1-]_input_highest*

Historical maximum power use

power[1-]_input_lowest*

Historical minimum power use.

power[1-]_reset_history*

Reset input_highest, input_lowest, average_highest and average_lowest.

power[1-]_accuracy*

Accuracy of the power meter.

power[1-]_cap*

If power use rises above this limit, the system should take action to reduce power use.

power[1-]_cap_hyst*

Margin of hysteresis built around capping and notification.

power[1-]_cap_max*

Maximum cap that can be set.

power[1-]_cap_min*

Minimum cap that can be set.

power[1-]_max*

Maximum power.

power[1-]_crit*

Critical maximum power.

If power rises to or above this limit, the system is expected take drastic action to reduce power consumption, such as a system shutdown or a forced powerdown of some devices.

Unit: microWatt

RW

power[1-]_enable*

Enable or disable the sensors.

When disabled the sensor read will return -ENODATA.

- 1: Enable
- 0: Disable

RW

power[1-]_rated_min*

Minimum rated power.

Unit: microWatt

RO

power[1-]_rated_max*

Maximum rated power.

Unit: microWatt

RO

Also see the Alarms section for status flags associated with power readings.

Energy

energy[1-]_input*

Cumulative energy use

Unit: microJoule

RO

energy[1-]_enable*

Enable or disable the sensors.

When disabled the sensor read will return -ENODATA.

- 1: Enable
- 0: Disable

RW

Humidity

humidity[1-]_input*
Humidity.
humidity[1-]_enable*
Enable or disable the sensors.
humidity[1-]_rated_min*
Minimum rated humidity.
humidity[1-]_rated_max*
Maximum rated humidity.

Alarms

Each channel or limit may have an associated alarm file, containing a boolean value. 1 means than an alarm condition exists, 0 means no alarm.

Usually a given chip will either use channel-related alarms, or limit-related alarms, not both. The driver should just reflect the hardware implementation.

`in[0-*]_alarm`, `curr[1-*]_alarm`, `power[1-*]_alarm`, `fan[1-*]_alarm`, `temp[1-*]_alarm`	Channel alarm • 0: no alarm • 1: alarm RO

OR

`in[0-*]_min_alarm`, `in[0-*]_max_alarm`, `in[0-*]_lcrit_alarm`, `in[0-*]_crit_alarm`, `curr[1-*]_min_alarm`, `curr[1-*]_max_alarm`, `curr[1-*]_lcrit_alarm`, `curr[1-*]_crit_alarm`, `power[1-*]_cap_alarm`, `power[1-*]_max_alarm`, `power[1-*]_crit_alarm`, `fan[1-*]_min_alarm`, `fan[1-*]_max_alarm`, `temp[1-*]_min_alarm`, `temp[1-*]_max_alarm`, `temp[1-*]_lcrit_alarm`, `temp[1-*]_crit_alarm`, `temp[1-*]_emergency_alarm`	Limit alarm • 0: no alarm • 1: alarm RO

Each input channel may have an associated fault file. This can be used to notify open diodes, unconnected fans etc. where the hardware supports it. When this boolean has value 1, the measurement for that channel should not be trusted.

fan[1-]_fault* / *temp[1-*]_fault*
Input fault condition.

Some chips also offer the possibility to get beeped when an alarm occurs:

beep_enable
Master beep enable.
in[0-]_beep*, *curr[1-*]_beep*, *fan[1-*]_beep*, *temp[1-*]_beep*,
Channel beep.

In theory, a chip could provide per-limit beep masking, but no such chip was seen so far.

Old drivers provided a different, non-standard interface to alarms and beeps. These interface files are deprecated, but will be kept around for compatibility reasons:

alarms
Alarm bitmask.
beep_mask
Bitmask for beep.

Intrusion detection

intrusion[0-]_alarm*
Chassis intrusion detection.

intrusion[0-]_beep*
: Chassis intrusion beep.

Average sample configuration

Devices allowing for reading {in,power,curr,temp}_average values may export attributes for controlling number of samples used to compute average.

samples	Sets number of average samples for all types of measurements. RW
in_samples power_samples curr_samples temp_samples	Sets number of average samples for specific type of measurements. Note that on some devices it won't be possible to set all of them to different values so changing one might also change some others. RW

sysfs attribute writes interpretation

hwmon sysfs attributes always contain numbers, so the first thing to do is to convert the input to a number, there are 2 ways todo this depending whether the number can be negative or not:

```
unsigned long u = simple_strtoul(buf, NULL, 10);
long s = simple_strtol(buf, NULL, 10);
```

With buf being the buffer with the user input being passed by the kernel. Notice that we do not use the second argument of strto[u]l, and thus cannot tell when 0 is returned, if this was really 0 or is caused by invalid input. This is done deliberately as checking this everywhere would add a lot of code to the kernel.

Notice that it is important to always store the converted value in an unsigned long or long, so that no wrap around can happen before any further checking.

After the input string is converted to an (unsigned) long, the value should be checked if its acceptable. Be careful with further conversions on the value before checking it for validity, as these conversions could still cause a wrap around before the check. For example do not multiply the result, and only add/subtract if it has been divided before the add/subtract.

What to do if a value is found to be invalid, depends on the type of the sysfs attribute that is being set. If it is a continuous setting like a tempX_max or inX_max attribute, then the value should be clamped to its limits using clamp_val(value, min_limit, max_limit). If it is not continuous like for example a tempX_type, then when an invalid value is written, -EINVAL should be returned.

Example1, temp1_max, register is a signed 8 bit value (-128 - 127 degrees):

```
long v = simple_strtol(buf, NULL, 10) / 1000;
v = clamp_val(v, -128, 127);
/* write v to register */
```

Example2, fan divider setting, valid values 2, 4 and 8:

```
unsigned long v = simple_strtoul(buf, NULL, 10);

switch (v) {
case 2: v = 1; break;
case 4: v = 2; break;
case 8: v = 3; break;
default:
        return -EINVAL;
}
/* write v to register */
```