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.

```
[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.

```
*********************
* Global attributes *
*********************

name		The chip name.
		This should be a short, lowercase string, not containing
		spaces nor dashes, representing the chip name. This is
		the only mandatory attribute.
		I2C devices get this attribute created automatically.
		RO

update_rate	The rate at which the chip will update readings.
		Unit: millisecond
		RW
		Some devices have a variable update rate. This attribute
		can be used to change the update rate to the desired
		frequency.


************
* Voltages *
************

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

in[0-*]_max	Voltage max value.
		Unit: millivolt
		RW
```

```
in[0-*]_input   Voltage input value.
                Unit: millivolt
                RO
                Voltage measured on the chip pin.
                Actual voltage depends on the scaling resistors on the
                motherboard, as recommended in the chip datasheet.
                This varies by chip and by motherboard.
                Because of this variation, values are generally NOT scaled
                by the chip driver, and must be done by the application.
                However, some drivers (notably lm87 and via686a)
                do scale, because of internal resistors built into a chip.
                These drivers will output the actual voltage. Rule of
                thumb: drivers should report the voltage values at the
                "pins" of the chip.

in[0-*]_label   Suggested voltage channel label.
                Text string
                Should only be created if the driver has hints about what
                this voltage channel is being used for, and user-space
                doesn't. In all other cases, the label is provided by
                user-space.
                RO

cpu[0-*]_vid    CPU core reference voltage.
                Unit: millivolt
                RO
                Not always correct.

vrm             Voltage Regulator Module version number.
                RW (but changing it should no more be necessary)
                Originally the VRM standard version multiplied by 10, but now
                an arbitrary number, as not all standards have a version
                number.
                Affects the way the driver calculates the CPU core reference
                voltage from the vid pins.

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


********
* Fans *
********

fan[1-*]_min    Fan minimum value
                Unit: revolution/min (RPM)
                RW

fan[1-*]_max    Fan maximum value
                Unit: revolution/min (RPM)
                Only rarely supported by the hardware.
                RW

fan[1-*]_input  Fan input value.
                Unit: revolution/min (RPM)
                RO
```

```
fan[1-*]_div    Fan divisor.
                Integer value in powers of two (1, 2, 4, 8, 16, 32, 64, 128).
                RW
                Some chips only support values 1, 2, 4 and 8.
                Note that this is actually an internal clock divisor, which
                affects the measurable speed range, not the read value.

fan[1-*]_target
                Desired fan speed
                Unit: revolution/min (RPM)
                RW
                Only makes sense if the chip supports closed-loop fan speed
                control based on the measured fan speed.

fan[1-*]_label  Suggested fan channel label.
                Text string
                Should only be created if the driver has hints about what
                this fan channel is being used for, and user-space doesn't.
                In all other cases, the label is provided by user-space.
                RO

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


*******
* PWM *
*******

pwm[1-*]        Pulse width modulation fan control.
                Integer value in the range 0 to 255
                RW
                255 is max or 100%.

pwm[1-*]_enable
                Fan speed control method:
                0: no fan speed control (i.e. fan at full speed)
                1: manual fan speed control enabled (using pwm[1-*])
                2+: automatic fan speed control enabled
                Check individual chip documentation files for automatic mode
                details.
                RW

pwm[1-*]_mode   0: DC mode (direct current)
                1: PWM mode (pulse-width modulation)
                RW

pwm[1-*]_freq   Base PWM frequency in Hz.
                Only possibly available when pwmN_mode is PWM, but not always
                present even then.
                RW

pwm[1-*]_auto_channels_temp
                Select which temperature channels affect this PWM output in
                auto mode. Bitfield, 1 is temp1, 2 is temp2, 4 is temp3 etc...
                Which values are possible depend on the chip used.
```

```
                RW

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. Number of trip points is
                chip-dependent. Use this for chips which associate trip points
                to PWM output channels.
                RW

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. Number of trip points is
                chip-dependent. Use this for chips which associate trip points
                to temperature channels.
                RW

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.
                Integers 1 to 6
                RW
                1: PII/Celeron Diode
                2: 3904 transistor
                3: thermal diode
                4: thermistor
                5: AMD AMDSI
                6: Intel PECI
                Not all types are supported by all chips

temp[1-*]_max   Temperature max value.
                Unit: millidegree Celsius (or millivolt, see below)
                RW

temp[1-*]_min   Temperature min value.
                Unit: millidegree Celsius
                RW

temp[1-*]_max_hyst
                Temperature hysteresis value for max limit.
                Unit: millidegree Celsius
                Must be reported as an absolute temperature, NOT a delta
                from the max value.
```

```
                RW

temp[1-*]_input Temperature input value.
                Unit: millidegree Celsius
                RO

temp[1-*]_crit  Temperature critical value, typically greater than
                corresponding temp_max values.
                Unit: millidegree Celsius
                RW

temp[1-*]_crit_hyst
                Temperature hysteresis value for critical limit.
                Unit: millidegree Celsius
                Must be reported as an absolute temperature, NOT a delta
                from the critical value.
                RW

temp[1-*]_offset
                Temperature offset which is added to the temperature reading
                by the chip.
                Unit: millidegree Celsius
                Read/Write value.

temp[1-*]_label Suggested temperature channel label.
                Text string
                Should only be created if the driver has hints about what
                this temperature channel is being used for, and user-space
                doesn't. In all other cases, the label is provided by
                user-space.
                RO

temp[1-*]_lowest
                Historical minimum temperature
                Unit: millidegree Celsius
                RO

temp[1-*]_highest
                Historical maximum temperature
                Unit: millidegree Celsius
                RO

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

temp_reset_history
                Reset temp_lowest and temp_highest for all sensors
                WO
```

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 *
************
```

Note that no known chip provides current measurements as of writing,
so this part is theoretical, so to say.

```
curr[1-*]_max   Current max value
                Unit: milliampere
                RW

curr[1-*]_min   Current min value.
                Unit: milliampere
                RW

curr[1-*]_input Current input value
                Unit: milliampere
                RO
```

```
*********
* Power *
*********

power[1-*]_average              Average power use
                                Unit: microWatt
                                RO

power[1-*]_average_interval     Power use averaging interval.  A poll
                                notification is sent to this file if the
                                hardware changes the averaging interval.
                                Unit: milliseconds
                                RW

power[1-*]_average_interval_max Maximum power use averaging interval
                                Unit: milliseconds
                                RO

power[1-*]_average_interval_min Minimum power use averaging interval
                                Unit: milliseconds
                                RO

power[1-*]_average_highest      Historical average maximum power use
                                Unit: microWatt
                                RO

power[1-*]_average_lowest       Historical average minimum power use
                                Unit: microWatt
                                RO

power[1-*]_average_max          A poll notification is sent to
```

power[1-*]_average when power use
rises above this value.
Unit: microWatt
RW

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

power[1-*]_input		Instantaneous power use
Unit: microWatt
RO

power[1-*]_input_highest	Historical maximum power use
Unit: microWatt
RO

power[1-*]_input_lowest		Historical minimum power use
Unit: microWatt
RO

power[1-*]_reset_history	Reset input_highest, input_lowest,
average_highest and average_lowest.
WO

power[1-*]_accuracy		Accuracy of the power meter.
Unit: Percent
RO

power[1-*]_alarm		1 if the system is drawing more power than the
cap allows; 0 otherwise. A poll notification is
sent to this file when the power use exceeds the
cap. This file only appears if the cap is known
to be enforced by hardware.
RO

power[1-*]_cap			If power use rises above this limit, the
system should take action to reduce power use.
A poll notification is sent to this file if the
cap is changed by the hardware. The *_cap
files only appear if the cap is known to be
enforced by hardware.
Unit: microWatt
RW

power[1-*]_cap_hyst		Margin of hysteresis built around capping and
notification.
Unit: microWatt
RW

power[1-*]_cap_max		Maximum cap that can be set.
Unit: microWatt
RO

```
power[1-*]_cap_min              Minimum cap that can be set.
                                Unit: microWatt
                                RO

**********
* Energy *
**********

energy[1-*]_input               Cumulative energy use
                                Unit: microJoule
                                RO


**********
* 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
fan[1-*]_alarm
temp[1-*]_alarm
                Channel alarm
                0: no alarm
                1: alarm
                RO

OR

in[0-*]_min_alarm
in[0-*]_max_alarm
fan[1-*]_min_alarm
fan[1-*]_max_alarm
temp[1-*]_min_alarm
temp[1-*]_max_alarm
temp[1-*]_crit_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.

in[0-*]_fault
fan[1-*]_fault
temp[1-*]_fault
                Input fault condition
                0: no fault occured
```

```
		1: fault condition
		RO

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

beep_enable	Master beep enable
		0: no beeps
		1: beeps
		RW

in[0-*]_beep
fan[1-*]_beep
temp[1-*]_beep
		Channel beep
		0: disable
		1: enable
		RW

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.
		RO
		Integer representation of one to four bytes.
		A '1' bit means an alarm.
		Chips should be programmed for 'comparator' mode so that
		the alarm will 'come back' after you read the register
		if it is still valid.
		Generally a direct representation of a chip's internal
		alarm registers; there is no standard for the position
		of individual bits. For this reason, the use of this
		interface file for new drivers is discouraged. Use
		individual *_alarm and *_fault files instead.
		Bits are defined in kernel/include/sensors.h.

beep_mask	Bitmask for beep.
		Same format as 'alarms' with the same bit locations,
		use discouraged for the same reason. Use individual
		*_beep files instead.
		RW


***********************
* Intrusion detection *
***********************

intrusion[0-*]_alarm
		Chassis intrusion detection
		0: OK
		1: intrusion detected
		RW
		Contrary to regular alarm flags which clear themselves
```

                automatically when read, this one sticks until cleared by
                the user. This is done by writing 0 to the file. Writing
                other values is unsupported.

intrusion[0-*]_beep
                Chassis intrusion beep
                0: disable
                1: enable
                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 SENSORS_LIMIT(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 = SENSORS_LIMIT(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 */
```