Linux power supply class

Synopsis

Power supply class used to represent battery, UPS, AC or DC power supply properties to user-space.

It defines core set of attributes, which should be applicable to (almost) every power supply out there. Attributes are available via sysfs and uevent interfaces.

Each attribute has well defined meaning, up to unit of measure used. While the attributes provided are believed to be universally applicable to any power supply, specific monitoring hardware may not be able to provide them all, so any of them may be skipped.

Power supply class is extensible, and allows to define drivers own attributes. The core attribute set is subject to the standard Linux evolution (i.e. if it will be found that some attribute is applicable to many power supply types or their drivers, it can be added to the core set).

It also integrates with LED framework, for the purpose of providing typically expected feedback of battery charging/fully charged status and AC/USB power supply online status. (Note that specific details of the indication (including whether to use it at all) are fully controllable by user and/or specific machine defaults, per design principles of LED framework).

Attributes/properties

Power supply class has predefined set of attributes, this eliminates code duplication across drivers. Power supply class insist on reusing its predefined attributes *and* their units.

So, userspace gets predictable set of attributes and their units for any kind of power supply, and can process/present them to a user in consistent manner. Results for different power supplies and machines are also directly comparable.

See drivers/power/supply/ds2760_battery.c and drivers/power/supply/pda_power.c for the example how to declare and handle attributes.

Units

Quoting include/linux/power_supply.h:

> All voltages, currents, charges, energies, time and temperatures in ÂµV, ÂµA, ÂµAh, ÂµWh, seconds and tenths of degree Celsius unless otherwise stated. It's driver's job to convert its raw values to units in which this class operates.

Attributes/properties detailed

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\power\[linux-master][Documentation][power]power_supply_class.rst, line 60)

Malformed table.

```
+------------------------------------------------------------------------+
|               **Charge/Energy/Capacity - how to not confuse**          |
+------------------------------------------------------------------------+
| **Because both "charge" (ÂµAh) and "energy" (ÂµWh) represents "capacity"  |
| of battery, this class distinguish these terms. Don't mix them!**      |
|                                                                        |
| - `CHARGE_*`                                                           |
|       attributes represents capacity in ÂµAh only.                      |
| - `ENERGY_*`                                                           |
|       attributes represents capacity in ÂµWh only.                      |
| - `CAPACITY`                                                           |
|       attribute represents capacity in *percents*, from 0 to 100.      |
+------------------------------------------------------------------------+
```

Postfixes:

_AVG
: *hardware* averaged value, use it if your hardware is really able to report averaged values.

_NOW
: momentary/instantaneous values.

STATUS
: this attribute represents operating status (charging, full, discharging (i.e. powering a load), etc.). This corresponds to *BATTERY_STATUS_** values, as defined in battery.h.

CHARGE_TYPE
: batteries can typically charge at different rates. This defines trickle and fast charges. For batteries that are already charged or discharging, 'n/a' can be displayed (or 'unknown', if the status is not known).

AUTHENTIC
: indicates the power supply (battery or charger) connected to the platform is authentic(1) or non authentic(0).

HEALTH
: represents health of the battery, values corresponds to POWER_SUPPLY_HEALTH_*, defined in battery.h.

VOLTAGE_OCV
: open circuit voltage of the battery.

VOLTAGE_MAX_DESIGN, VOLTAGE_MIN_DESIGN
: design values for maximal and minimal power supply voltages. Maximal/minimal means values of voltages when battery considered "full"/"empty" at normal conditions. Yes, there is no direct relation between voltage and battery capacity, but some dumb batteries use voltage for very approximated calculation of capacity. Battery driver also can use this attribute just to inform userspace about maximal and minimal voltage thresholds of a given battery.

VOLTAGE_MAX, VOLTAGE_MIN
: same as _DESIGN voltage values except that these ones should be used if hardware could only guess (measure and retain) the thresholds of a given power supply.

VOLTAGE_BOOT
: Reports the voltage measured during boot

CURRENT_BOOT
: Reports the current measured during boot

CHARGE_FULL_DESIGN, CHARGE_EMPTY_DESIGN
: design charge values, when battery considered full/empty.

ENERGY_FULL_DESIGN, ENERGY_EMPTY_DESIGN
: same as above but for energy.

CHARGE_FULL, CHARGE_EMPTY
: These attributes means "last remembered value of charge when battery became full/empty". It also could mean "value of charge when battery considered full/empty at given conditions (temperature, age)". I.e. these attributes represents real thresholds, not design values.

ENERGY_FULL, ENERGY_EMPTY
: same as above but for energy.

CHARGE_COUNTER
: the current charge counter (in ÂµAh). This could easily be negative; there is no empty or full value. It is only useful for relative, time-based measurements.

PRECHARGE_CURRENT
: the maximum charge current during precharge phase of charge cycle (typically 20% of battery capacity).

CHARGE_TERM_CURRENT
: Charge termination current. The charge cycle terminates when battery voltage is above recharge threshold, and charge current is below this setting (typically 10% of battery capacity).

CONSTANT_CHARGE_CURRENT
: constant charge current programmed by charger.

CONSTANT_CHARGE_CURRENT_MAX
: maximum charge current supported by the power supply object.

CONSTANT_CHARGE_VOLTAGE
: constant charge voltage programmed by charger.

CONSTANT_CHARGE_VOLTAGE_MAX
: maximum charge voltage supported by the power supply object.

INPUT_CURRENT_LIMIT
: input current limit programmed by charger. Indicates the current drawn from a charging source.

INPUT_VOLTAGE_LIMIT
: input voltage limit programmed by charger. Indicates the voltage limit from a charging source.

INPUT_POWER_LIMIT
: input power limit programmed by charger. Indicates the power limit from a charging source.

CHARGE_CONTROL_LIMIT
: current charge control limit setting

CHARGE_CONTROL_LIMIT_MAX
: maximum charge control limit setting

CALIBRATE
: battery or coulomb counter calibration status

CAPACITY
: capacity in percents.

CAPACITY_ALERT_MIN
: minimum capacity alert value in percents.

CAPACITY_ALERT_MAX
: maximum capacity alert value in percents.

CAPACITY_LEVEL
: capacity level. This corresponds to POWER_SUPPLY_CAPACITY_LEVEL_*.

TEMP
: temperature of the power supply.

TEMP_ALERT_MIN
: minimum battery temperature alert.

TEMP_ALERT_MAX
: maximum battery temperature alert.

TEMP_AMBIENT
: ambient temperature.

TEMP_AMBIENT_ALERT_MIN
: minimum ambient temperature alert.

TEMP_AMBIENT_ALERT_MAX
: maximum ambient temperature alert.

TEMP_MIN
: minimum operatable temperature

TEMP_MAX
: maximum operatable temperature

TIME_TO_EMPTY
: seconds left for battery to be considered empty (i.e. while battery powers a load)

TIME_TO_FULL
: seconds left for battery to be considered full (i.e. while battery is charging)

Battery <-> external power supply interaction

Often power supplies are acting as supplies and supplicants at the same time. Batteries are good example. So, batteries usually care if they're externally powered or not.

For that case, power supply class implements notification mechanism for batteries.

External power supply (AC) lists supplicants (batteries) names in "supplied_to" struct member, and each power_supply_changed() call issued by external power supply will notify supplicants via external_power_changed callback.

Devicetree battery characteristics

Drivers should call power_supply_get_battery_info() to obtain battery characteristics from a devicetree battery node, defined in Documentation/devicetree/bindings/power/supply/battery.yaml. This is implemented in drivers/power/supply/bq27xxx_battery.c.

Properties in struct power_supply_battery_info and their counterparts in the battery node have names corresponding to elements in enum power_supply_property, for naming consistency between sysfs attributes and battery node properties.

QA

Q:
: Where is POWER_SUPPLY_PROP_XYZ attribute?

A:
: If you cannot find attribute suitable for your driver needs, feel free to add it and send patch along with your driver.

  The attributes available currently are the ones currently provided by the drivers written.

  Good candidates to add in future: model/part#, cycle_time, manufacturer, etc.

Q:
: I have some very specific attribute (e.g. battery color), should I add this attribute to standard ones?

A:
: Most likely, no. Such attribute can be placed in the driver itself, if it is useful. Of course, if the attribute in question applicable to large set of batteries, provided by many drivers, and/or comes from some general battery specification/standard, it may be a candidate to be added to the core attribute set.

Q:
: Suppose, my battery monitoring chip/firmware does not provides capacity in percents, but provides charge_{now,full,empty}. Should I calculate percentage capacity manually, inside the driver, and register CAPACITY attribute? The same question about time_to_empty/time_to_full.

A:
: Most likely, no. This class is designed to export properties which are directly measurable by the specific hardware available.

Inferring not available properties using some heuristics or mathematical model is not subject of work for a battery driver. Such functionality should be factored out, and in fact, apm_power, the driver to serve legacy APM API on top of power supply class, uses a simple heuristic of approximating remaining battery capacity based on its charge, current, voltage and so on. But full-fledged battery model is likely not subject for kernel at all, as it would require floating point calculation to deal with things like differential equations and Kalman filters. This is better be handled by batteryd/libbattery, yet to be written.