# Regulator Consumer Driver Interface

This text describes the regulator interface for consumer device drivers. Please see overview.txt for a description of the terms used in this text.

## 1. Consumer Regulator Access (static & dynamic drivers)

A consumer driver can get access to its supply regulator by calling

```
regulator = regulator_get(dev, "Vcc");
```

The consumer passes in its struct device pointer and power supply ID. The core then finds the correct regulator by consulting a machine specific lookup table. If the lookup is successful then this call will return a pointer to the struct regulator that supplies this consumer.

To release the regulator the consumer driver should call

```
regulator_put(regulator);
```

Consumers can be supplied by more than one regulator e.g. codec consumer with analog and digital supplies

```
digital = regulator_get(dev, "Vcc");  /* digital core */
analog = regulator_get(dev, "Avdd");  /* analog */
```

The regulator access functions regulator_get() and regulator_put() will usually be called in your device drivers probe() and remove() respectively.

## 2. Regulator Output Enable & Disable (static & dynamic drivers)

A consumer can enable its power supply by calling:

```
int regulator_enable(regulator);
```

NOTE:
: The supply may already be enabled before regulator_enabled() is called. This may happen if the consumer shares the regulator or the regulator has been previously enabled by bootloader or kernel board initialization code.

A consumer can determine if a regulator is enabled by calling:

```
int regulator_is_enabled(regulator);
```

This will return > zero when the regulator is enabled.

A consumer can disable its supply when no longer needed by calling:

```
int regulator_disable(regulator);
```

NOTE:
: This may not disable the supply if it's shared with other consumers. The regulator will only be disabled when the enabled reference count is zero.

Finally, a regulator can be forcefully disabled in the case of an emergency:

```
int regulator_force_disable(regulator);
```

NOTE:
: this will immediately and forcefully shutdown the regulator output. All consumers will be powered off.

## 3. Regulator Voltage Control & Status (dynamic drivers)

Some consumer drivers need to be able to dynamically change their supply voltage to match system operating points. e.g. CPUfreq drivers can scale voltage along with frequency to save power, SD drivers may need to select the correct card voltage, etc.

Consumers can control their supply voltage by calling:

```
int regulator_set_voltage(regulator, min_uV, max_uV);
```

Where min_uV and max_uV are the minimum and maximum acceptable voltages in microvolts.

NOTE: this can be called when the regulator is enabled or disabled. If called when enabled, then the voltage changes instantly, otherwise the voltage configuration changes and the voltage is physically set when the regulator is next enabled.

The regulators configured voltage output can be found by calling:

```
int regulator_get_voltage(regulator);
```

NOTE:
get_voltage() will return the configured output voltage whether the regulator is enabled or disabled and should NOT be used to determine regulator output state. However this can be used in conjunction with is_enabled() to determine the regulator physical output voltage.

## 4. Regulator Current Limit Control & Status (dynamic drivers)

Some consumer drivers need to be able to dynamically change their supply current limit to match system operating points. e.g. LCD backlight driver can change the current limit to vary the backlight brightness, USB drivers may want to set the limit to 500mA when supplying power.

Consumers can control their supply current limit by calling:

```
int regulator_set_current_limit(regulator, min_uA, max_uA);
```

Where min_uA and max_uA are the minimum and maximum acceptable current limit in microamps.

NOTE:
this can be called when the regulator is enabled or disabled. If called when enabled, then the current limit changes instantly, otherwise the current limit configuration changes and the current limit is physically set when the regulator is next enabled.

A regulators current limit can be found by calling:

```
int regulator_get_current_limit(regulator);
```

NOTE:
get_current_limit() will return the current limit whether the regulator is enabled or disabled and should not be used to determine regulator current load.

## 5. Regulator Operating Mode Control & Status (dynamic drivers)

Some consumers can further save system power by changing the operating mode of their supply regulator to be more efficient when the consumers operating state changes. e.g. consumer driver is idle and subsequently draws less current

Regulator operating mode can be changed indirectly or directly.

### Indirect operating mode control.

Consumer drivers can request a change in their supply regulator operating mode by calling:

```
int regulator_set_load(struct regulator *regulator, int load_uA);
```

This will cause the core to recalculate the total load on the regulator (based on all its consumers) and change operating mode (if necessary and permitted) to best match the current operating load.

The load_uA value can be determined from the consumer's datasheet. e.g. most datasheets have tables showing the maximum current consumed in certain situations.

Most consumers will use indirect operating mode control since they have no knowledge of the regulator or whether the regulator is shared with other consumers.

### Direct operating mode control.

Bespoke or tightly coupled drivers may want to directly control regulator operating mode depending on their operating point. This can be achieved by calling:

```
int regulator_set_mode(struct regulator *regulator, unsigned int mode);
unsigned int regulator_get_mode(struct regulator *regulator);
```

Direct mode will only be used by consumers that *know* about the regulator and are not sharing the regulator with other consumers.

## 6. Regulator Events

Regulators can notify consumers of external events. Events could be received by consumers under regulator stress or failure conditions.

Consumers can register interest in regulator events by calling:

```
int regulator_register_notifier(struct regulator *regulator,
                                struct notifier_block *nb);
```

Consumers can unregister interest by calling:

```
int regulator_unregister_notifier(struct regulator *regulator,
                                  struct notifier_block *nb);
```

Regulators use the kernel notifier framework to send event to their interested consumers.

## 7. Regulator Direct Register Access

Some kinds of power management hardware or firmware are designed such that they need to do low-level hardware access to regulators, with no involvement from the kernel. Examples of such devices are:

- clocksource with a voltage-controlled oscillator and control logic to change the supply voltage over I2C to achieve a desired output clock rate
- thermal management firmware that can issue an arbitrary I2C transaction to perform system poweroff during overtemperature conditions

To set up such a device/firmware, various parameters like I2C address of the regulator, addresses of various regulator registers etc. need to be configured to it. The regulator framework provides the following helpers for querying these details.

Bus-specific details, like I2C addresses or transfer rates are handled by the regmap framework. To get the regulator's regmap (if supported), use:

```
struct regmap *regulator_get_regmap(struct regulator *regulator);
```

To obtain the hardware register offset and bitmask for the regulator's voltage selector register, use:

```
int regulator_get_hardware_vsel_register(struct regulator *regulator,
                                         unsigned *vsel_reg,
                                         unsigned *vsel_mask);
```

To convert a regulator framework voltage selector code (used by regulator_list_voltage) to a hardware-specific voltage selector that can be directly written to the voltage selector register, use:

```
int regulator_list_hardware_vsel(struct regulator *regulator,
                                 unsigned selector);
```