

# Linux wireless regulatory documentation

This document gives a brief review over how the Linux wireless regulatory infrastructure works.

More up to date information can be obtained at the project's web page:

<https://wireless.wiki.kernel.org/en/developers/Regulatory>

## Keeping regulatory domains in userspace

Due to the dynamic nature of regulatory domains we keep them in userspace and provide a framework for userspace to upload to the kernel one regulatory domain to be used as the central core regulatory domain all wireless devices should adhere to.

## How to get regulatory domains to the kernel

When the regulatory domain is first set up, the kernel will request a database file (regulatory.db) containing all the regulatory rules. It will then use that database when it needs to look up the rules for a given country.

## How to get regulatory domains to the kernel (old CRDA solution)

Userspace gets a regulatory domain in the kernel by having a userspace agent build it and send it via nl80211. Only expected regulatory domains will be respected by the kernel.

A currently available userspace agent which can accomplish this is CRDA - central regulatory domain agent. Its documented here:

<https://wireless.wiki.kernel.org/en/developers/Regulatory/CRDA>

Essentially the kernel will send a udev event when it knows it needs a new regulatory domain. A udev rule can be put in place to trigger crda to send the respective regulatory domain for a specific ISO/IEC 3166 alpha2.

Below is an example udev rule which can be used:

```
# Example file, should be put in /etc/udev/rules.d/regulatory.rules KERNEL=="regulatory*", ACTION=="change",  
SUBSYSTEM=="platform", RUN+="/sbin/crda"
```

The alpha2 is passed as an environment variable under the variable COUNTRY.

## Who asks for regulatory domains?

- Users

Users can use iw:

<https://wireless.wiki.kernel.org/en/users/Documentation/iw>

An example:

```
# set regulatory domain to "Costa Rica"  
iw reg set CR
```

This will request the kernel to set the regulatory domain to the specified alpha2. The kernel in turn will then ask userspace to provide a regulatory domain for the alpha2 specified by the user by sending a uevent.

- Wireless subsystems for Country Information elements

The kernel will send a uevent to inform userspace a new regulatory domain is required. More on this to be added as its integration is added.

- Drivers

If drivers determine they need a specific regulatory domain set they can inform the wireless core using `regulatory_hint()`. They have two options -- they either provide an alpha2 so that crda can provide back a regulatory domain for that country or they can build their own regulatory domain based on internal custom knowledge so the wireless core can respect it.

Most drivers will rely on the first mechanism of providing a regulatory hint with an alpha2. For these drivers there is an additional check that can be used to ensure compliance based on custom EEPROM regulatory data. This additional check can be used by drivers by registering on its struct `wiphy` a `reg_notifier()` callback. This notifier is called when the core's regulatory domain has been changed. The driver can use this to review the changes made and also review who made them (driver, user, country IE) and determine what to allow based on its internal EEPROM data. Devices drivers wishing to be capable of world roaming should use this callback. More on world roaming will be added to this document when its support is enabled.

Device drivers who provide their own built regulatory domain do not need a callback as the channels registered by them are the only ones that will be allowed and therefore *additional* channels cannot be enabled.

## Example code - drivers hinting an alpha2:

This example comes from the `zd1211rw` device driver. You can start by having a mapping of your device's EEPROM country/regulatory domain value to a specific alpha2 as follows:

```
static struct zd_reg_alpha2_map reg_alpha2_map[] = {
    { ZD_REGDOMAIN_FCC, "US" },
    { ZD_REGDOMAIN_IC, "CA" },
    { ZD_REGDOMAIN_ETSI, "DE" }, /* Generic ETSI, use most restrictive */
    { ZD_REGDOMAIN_JAPAN, "JP" },
    { ZD_REGDOMAIN_JAPAN_ADD, "JP" },
    { ZD_REGDOMAIN_SPAIN, "ES" },
    { ZD_REGDOMAIN_FRANCE, "FR" },
}
```

Then you can define a routine to map your read EEPROM value to an alpha2, as follows:

```
static int zd_reg2alpha2(u8 regdomain, char *alpha2)
{
    unsigned int i;
    struct zd_reg_alpha2_map *reg_map;
    for (i = 0; i < ARRAY_SIZE(reg_alpha2_map); i++) {
        reg_map = &reg_alpha2_map[i];
        if (regdomain == reg_map->reg) {
            alpha2[0] = reg_map->alpha2[0];
            alpha2[1] = reg_map->alpha2[1];
            return 0;
        }
    }
    return 1;
}
```

Lastly, you can then hint to the core of your discovered alpha2, if a match was found. You need to do this after you have registered your wiphy. You are expected to do this during initialization.

```
r = zd_reg2alpha2(mac->regdomain, alpha2);
if (!r)
    regulatory_hint(hw->wiphy, alpha2);
```

## Example code - drivers providing a built in regulatory domain:

[NOTE: This API is not currently available, it can be added when required]

If you have regulatory information you can obtain from your driver and you *need* to use this we let you build a regulatory domain structure and pass it to the wireless core. To do this you should `kmalloc()` a structure big enough to hold your regulatory domain structure and you should then fill it with your data. Finally you simply call `regulatory_hint()` with the regulatory domain structure in it.

Bellow is a simple example, with a regulatory domain cached using the stack. Your implementation may vary (read EEPROM cache instead, for example).

Example cache of some regulatory domain:

```
struct ieee80211_regdomain mydriver_jp_regdom = {
    .n_reg_rules = 3,
    .alpha2 = "JP",
    //.alpha2 = "99", /* If I have no alpha2 to map it to */
    .reg_rules = {
        /* IEEE 802.11b/g, channels 1..14 */
        REG_RULE(2412-10, 2484+10, 40, 6, 20, 0),
        /* IEEE 802.11a, channels 34..48 */
        REG_RULE(5170-10, 5240+10, 40, 6, 20,
            NL80211_RRF_NO_IR),
        /* IEEE 802.11a, channels 52..64 */
        REG_RULE(5260-10, 5320+10, 40, 6, 20,
            NL80211_RRF_NO_IR|
            NL80211_RRF_DFS),
    }
};
```

Then in some part of your code after your wiphy has been registered:

```
struct ieee80211_regdomain *rd;
int size_of_regd;
int num_rules = mydriver_jp_regdom.n_reg_rules;
unsigned int i;

size_of_regd = sizeof(struct ieee80211_regdomain) +
    (num_rules * sizeof(struct ieee80211_reg_rule));
```

```
rd = kzalloc(size_of_regd, GFP_KERNEL);
if (!rd)
    return -ENOMEM;

memcpy(rd, &mydriver_jp_regdom, sizeof(struct ieee80211_regdomain));

for (i=0; i < num_rules; i++)
    memcpy(&rd->reg_rules[i],
           &mydriver_jp_regdom.reg_rules[i],
           sizeof(struct ieee80211_reg_rule));
regulatory_struct_hint(rd);
```

## Statically compiled regulatory database

When a database should be fixed into the kernel, it can be provided as a firmware file at build time that is then linked into the kernel.